1982

An investigation of the feasibility of implementing substantial finite element codes on popular microcomputers.

Mulholland, David Joseph.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN INVESTIGATION OF THE FEASIBILITY OF IMPLEMENTING SUBSTANTIAL FINITE ELEMENT CODES ON POPULAR MICROCOMPUTERS

by

David Joseph Mulholland

October 1982

Thesis Advisor: G. Cantin

Approved for public release; distribution unlimited.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): An Investigation of the Feasibility of Implementing Substantial Finite Element Codes on Popular Microcomputers	5. TYPE OF REPORT & PERIOD COVERED: Masters Thesis; October 1982
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): David Joseph Mulholland	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93940	12. REPORT DATE: October 1982
	13. NUMBER OF PAGES: 287
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Microcomputers
Finite Elements
Trusses
Linear Equation Solvers

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The size and cost of microcomputers continue to decrease while their memory capacity and execution speed increase. These advances should result in small, inexpensive machines attaining the same computing power as current mainframe models. The interim need is to adapt general finite element codes to present day, less capable microcomputers. This thesis explores the program structure, memory management, I/O procedures and

equation solving methods necessary to accomplish that task. The equation solving capacity and speed of the Apple-II Plus Personal Computer System and the Hewlett-Packard System 45(A) Desktop Computer are compared. A finite element program for the static analysis of space trusses is presented, as adapated to and tested on the Apple-II Plus. The program output may be printed in either English or French.

Approved for public release; distribution unlimited.

An Investigation of the Feasibility of Implementing Substantial Finite Element Codes on Popular Microcomputers

by

David Joseph Mulholland
Lieutenant, United States Navy
B.S.M.E., University of Utah, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
October 1982

ABSTRACT

The size and cost of microcomputers continue to decrease while their memory capacity and execution speed increase. These advances should result in small, inexpensive machines attaining the same computing power as current mainframe models. The interim need is to adapt general finite element codes to present day, less capable microcomputers. This thesis explores the program structure, memory management, I/O procedures and equation solving methods necessary to accomplish that task. The equation solving capacity and speed of the Apple-II Plus Personal Computer Systems and the Hewlett-Packard System 45(A) Desktop Computer are compared. A finite element program for the static analysis of space trusses is presented, as adapted to and tested on the Apple-II Plus. The program output may be printed in either English or French.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

The author wishes to express his sincere appreciation to Professor Gilles Cantin of the Naval Postgraduate School for suggesting the original idea for this thesis and for providing the technical guidance and personal encouragement necessary for its successful completion.

The author also wishes to thank Dr. W. M. Woods, Dean of Educational Development, and Paul W. Sparks, Electronics Technician, Operations Research Department, both of the Naval Postgraduate School, for most generously donating their time and the equipment necessary for the testing of the Apple-II Plus Personal Computer System.

Lastly, the author wishes to acknowledge the support of his wife Mary, whose patience during their many hours of separation, and forbearance while his attentions were so often lavished on "the computer," have made the completion of this work possible.

I. INTRODUCTION

A. PURPOSE AND SCOPE OF THE INVESTIGATION

In the preface of the third edition of his classic reference on the finite element method, Zienkiewicz [1] made an assessment of the "state-of-the-art" of this most useful analytical technique. In 1977, Zienkiewicz described a discipline which had ammassed a collection of nearly 8000 published reference works in an ever widening variety of applications, and which had evolved far beyond its roots as an engineering tool for linear analysis of solid mechanics structures. Present applications commonly include heat transfer, fluid mechanics and non-linear solid mechanics, as well as the complete static and dynamic analysis of engineering structures.

A more geneal statement may be made [1] in describing the finite element method as "... a general discretization procedure of continuum problems, posed by mathematically defined statements." The method now becomes the means of approximating a continuous problem with a discrete model composed of finite elements, as opposed to "infinitesimal" elements in the sense of the calculus. In some cases, in the limit, as the number of elements increases to infinity and the physical dimensions these elements represent decrease, the solution of the model becomes exact. It is this generality of method

and application which has been one of the two most important factors influencing the widespread use of the method in scientific research and industrial applications.

The second factor influencing the spread of the finite element method has been the availability of high speed, high capacity, general purpose digital computers. It is well known that the method remained in its infancy until general purpose digital computers became available in reasonable numbers. Today technological advances in microelectronics and solid state devices have drastically reduced the real cost of computer time. Advances in the art of building static, random access memory now allow the use of Direct Memory Access techniques in small (16+ bit word) desktop machines. When these machines are interfaced with magnetic hard disk drives, a small user easily has the potential of 16 Megabytes (8 binary bits equals one byte) of high speed storage, at a small fraction of the cost of a large main-frame computer [2]. These developments are truely revolutionary. Very "friendly" and general structural finite element codes, some with appealing interactive graphics capabilities, may soon become available to the smallest of organizations.

In addition to the new machines now on the horizon, there are a very large number of smaller, less capable computers presently in service. These smaller machines, often described as "microcomputers," vary in word length and storage

capacity, generally having from 16 to 256 thousand bytes of random access memory. They support a wide variety of peripheral equipments, the most indispensible of which are high speed, magnetic disk drives.[1] These smaller machines are proportionately less expensive than larger computers, which accounts for the very great demand for these systems in industry, government and the private sector. Many of the most popular models began as "personal" or leisure computers for hobbiests, but rapidly moved into small business, medical and financial management applications as the utility of the machines was realized and software became available.

If one looks, however, for finite element software or attempts to find evidence of this widespread analytical technique being implemented in any substantial manner on microcomputers, the results are disappointing. It is the thesis of this investigation that:

(1) Substantial problems in finite element analysis can be solved on microcomputers.

(2) Microcomputers can solve substantial finite element problems with acceptable accuracy and to reasonable precision.

---

[1] Disk storage extends the usefulness of microcomputers by avoiding or softening the impact of some of the limits imposed by having too little main "core" memory.

(3) Although the central processor clock speed of most popular microcomputers is much less than that of a mainframe computer, and in spite of the interpretive nature of most microcomputer languages, substantial finite element problems can be solved on these machines in an acceptable amount of time.

(4) General guidelines can be established for using innovative program structuring, memory management, software overlay schemes, data base management and high speed, disk storage to allow the implementation of substantial finite element codes on small popular microcomputers.

B. CHOICE OF MACHINES AND PERIPHERAL DEVICES

The choice of machines was influenced most heavily by the types of systems available at the Naval Postgraduate School. A Hewlett-Packard System 45 Desktop Computer was available within the Mechanical Engineering Department and was a logical first choice. Previous work had already been done on implementing a finite element code on this system [3] and the potential existed for increasing the capabilities of the program by adding additional finite elements, changing from an "in-core" to an "out-of-core" equation solver, and using magnetic floppy disk instead of magnetic cartridge tape mass storage. These systems are not uncommon within the Department of Defense and the Navy and the potential benefit from an improved program was judged to be worth the additional effort which would be required to upgrade the code.

The System 45 is, however, nearly outside the boundaries of the group of machines previously described as "popular microcomputers." The base price of the computer alone, exclusive of the disk drives, was over $29,000.00 in 1977. A new System 45 (HP 9845C), upgraded and having slightly more computing capability, would have cost about $39,500.00 if purchased in 1980 [4] and still be without disk mass storage. It was decided, therefore, to also choose a system which would represent those on the lower end of the cost spectrum. Several Apple-II Plus[2] Personal Computer Systems were available, having a total price, including multiple disk drives, of approximately $6,000.00. In addition to having a low cost, the Apple-II Plus systems also had the capability of the FORTRAN Language Option, not available on the HP 9845A.

For the reasons enumerated above, it was decided to investigate the equation solving capability of the Apple-II Plus Personal Computer and the Hewlett-Packard System 45 Desktop Computer (HP 9845A) and to attempt, should adequate time be available, the installation of representative finite element codes on both systems.

1. The Hewlett Packard 9845 Desktop Computer

The System 45 was, in 1980, the top-of-the-line model of Hewlett-Packard desktop computers. As with all of this

---

[2]Apple-II Plus is a registered trademark of the Apple Computer Company.

firm's computers, technical details about the hardware are closely held by the company and few are available in users manuals supplied with the machine or in the Hewlett-Packard Journal. References [5] and [6] and the installation documents for the system provided the information necessary to conduct the investigation.

Only a very brief summary of the most important details appears in the following sections, and readers intending to use programs presented in this thesis must become familiar with the machine in general and their own systems in particular.

a. System Configuration

The system configuration during the investigation is shown in the table below. Only the minimum devices and options which are required to run the program presented in this thesis are shown. Note that the possession of a more capable system than the one shown (e.g. one hard disk instead of dual flexible disk drives) will not preclude use of the proposed software. The reader need only change the "mass storage unit specifier" [6] and "select" [5] codes to match his/her particular configuration.

Although, under the "unified mass storage concept" [5], it is possible for a user to substitute cartridge tape for disk mass storage in nearly any program, this practice is unsuitable for solution of large systems of linear equations out-of-core. Previous experience with less demanding mass storage tasks [4] showed degradation of system

performance, on large problems, where tapes were required to rewind too many times. Stretching of the tape initially introduced parity errors and later caused breakage of the tape.

Table 1: Configuration of the Hewlett-Packard Systems 45 Installation

Required Devices:

Part No.	Description	Remarks
HP 9845A	Computer with CRT	
HP 98032A	16 Bit Interface (9885A Disk)	Option 85
HP 9885M	Flexible Disk Drive (Master)	Drive 0
HP 9885S	Flexible Disk Drive (Slave)	Drive 1

Required Options:

Number	Description	Remarks
203	64K RAM	
310	Mass Storage ROM	
320	I/O ROM Left	
330	I/O ROM Right	
370	Graphics ROM	Select Code 13
500	Internal Thermal Printer	Select Code 0

Select Code Summary:

Code	Device
0	Printer
8	Flexible Disk Drive (Controller)
13	Graphics ROM
16	CRT Display

b. Machine Precision

The HP9845A stores floating point numbers between $9.99999999999 \times 10^{99}$ and 1×10^{-99} in absolute value. An extended calculation range of $9.99999999999 \times 10^{511}$ to 1×10^{-511} is used for the results of intermediate calculations [5]. Floating point numbers are described as full precision or short precision. Full precision floating point variables each occupy 8 bytes of storage and are used exclusively in all HP Enhanced BASIC Programs presented in this thesis.

Chapter 3 of Reference [7] discussed the estimation of machine roundoff error (as part of a discussion on estimating the relative error in the solution of systems of linear equations from the matrix condition number). The worst case roundoff error, in base 10 for a 12 digit machine is:

$$10^{-12+1} = 0.00000000001$$

Section 2.2 of Reference [7] also defines a quantity called the machine epsilon as the smallest floating point number distinguishable from zero. A program, modeled after that presented in the reference, found the machine epsilon of the HP9845 to be 7.27595761415E-12.

c. The Enhanced BASIC Language

The Enhanced BASIC Language from Hewlett-Packard has all the features of the most powerful, high level versions

of BASIC (Beginners All-purpose Symbolic Instruction Code) and certain extensions designed to provide optimum use of the System 45. References [5] and [6] contain a complete description of the language and references for further study.

As with most versions of microcomputer BASIC, the language is completely interpretive in nature. That is, no compilation or conversion of source code to machine language is accomplished before executing the program. Each line of source code is interpreted, line-by-line, as execution progresses. Although this makes it possible for the program to have such incredible flexibility as to be able, in some cases, to edit itself during execution, it also considerably slows the speed of execution. On the positive side, it is true that there are many instructions in Hewlett-Packard Enhanced BASIC which can reduce the amount of source code required over, say, the same program in FORTRAN (e.g., the MAT series of instructions for matrix manipulation [5]).

The reader who is familiar with FORTRAN will have little trouble following the programs presented in this thesis. The table below lists some of the more unusual features of the language and should be enough to allow complete understanding of program flow. For further information, the reader must consult the references given above.

Table 2: Selected Members of the Hewlett-Packard Enhanced BASIC Instruction Set

Definitions:

"filenumber" is a numeral or numeric expression having an integer value between 1 and 10.

"file specifier" is a character string enclosed in quotation marks or a string variable which is a valid and complete file name (including mass storage unit specifier).

Statements:

ASSIGN# filenumber TO file specifier
The ASSIGN statement places the filename in an internal table and allows the file to be referenced by a single integer number in PRINT# or READ# statements [5].

BUFFER# filenumber
Sets up an additional 256 byte I/O buffer for all READ# and PRINT# statements to the specified filenumber, allowing the most efficient coupling between a file and its associated mass storage device [6].

OPTION BASE 1
Specifies that the lower bound of all array variable subscripts shall be 1 (e.g. Vector (1) is allowed but Vector (0) is not) [5].

Yyy = FNXxxxxx (aa, B,Ccc,...)
Call to a function subprogram previously defined by a DEF FNXxxxxx statement [5]. In FORTRAN this would be YYY = XXXXXX (AA,B,CCC,...) where XXXXXX is a FUNCTION subprogram or an intrinsic function set up with a statement function definition.

Zzz000:
An example of a label to which program flow may be transferred and which keeps its relative position within the file regardless of changes in line numbering [5]. Any proper statement may follow the colon.

BEEP
Causes a tone to be sounded at the keyboard [5].

DISP "Character string"
Causes the character string enclosed in quotation marks to be printed in the display area of the CRT [5].

OVERLAP
Allows computation and I/O to occur simultaneously. Most beneficial when a program has nearly equal amounts of computation and I/O. This mode is disabled by the SERIAL statement. See [5] and its references for a discussion of the OVERLAP statements impact on error traps.

When attempting to reduce the run time of a program, it is important to recall the method used by the HP9845 when a subroutine or function call, or transfer to a statement label or line number, is encountered. The BASIC interpreter jumps to the beginning of the file whenever a search for one of these items is instituted. Run speed can be improved significantly when the most frequently used functions and subroutines are placed as near to the beginning of a long program as possible. Note also that comment lines in BASIC (statements preceeded by REM or an exclamation point) are now a liability because they increase the length of a program. The amount of documentary comments should be limited to the minimum necessary to understand the code.

It is possible to overlay subroutines, by using the LINK statement, at any line number in the main program (or in place of the main program if desired) [5]. Overlaying reduces the amount of program in core, allowing the storage of more variables or the execution of programs too large to fit in memory in a single piece.

d. Disk Mass Storage Considerations

The characteristics of the disk mass storage hardware used with any microcomputer have a significant impact on the performance of the system in finite element calculations. The size and number of usable physical records on the disk, size of the file directory available and types of file structure and access allowed are all important considerations in attaining optimum use of out of core storage.

In the HP 9845 there are two types of records with which the programmer must be concerned [6]. A physical record is the unit of storage dealt with by the mass storage device itself (and is usually the buffer size associated with the device). Defined records are the smallest addressable units of storage, in even numbers of bytes, which can be individually accessed by the user. Whenever possible the defined record length should match the physical record length of the device in use. In the HP 9845 this matching of record lengths can provide an improvement in I/O performance of two- or three-to-one.

Reference [6] describes the use of binary DATA files and Direct Memory Access (data transfer without the use of buffers) for the rapid transfer of entire arrays between disk mass storage and core memory. This method can result in a considerable I/O improvement over buffering techniques, depending on the storage device and the amount of data to be accomplished.

The characteristics of the HP 9885 Flexible Disk Drive are listed in the table below.

Table 3: Selected Characteristics of the Hewlett-Packard 9885 Flexible Disk Drive

Storage Capacity:	
Bytes (total)	499,200
Physical records (maximum)	1,950
Physical records (useable)	1,901
Files (Director size limit)	352
Bytes per physical record	256
Accessing:	
Maximum transfer rate (bytes per second)	46,000

2. The Apple-II Plus Personal Computer

The Apple-II Plus Personal Computer is manufactured by Apple Computer Incorporated of Cupertino, California and is designed around the MOS Technology 6502 microprocessor. The maximum addressing range is 2^{16} (64K) locations on a 16 bit parallel bus. The data bus is 8 bits, parallel and bi-directional and the CPU clock speed is 1.203 MHz. More technical information is available in Reference [8].

a. System Configuration

The baseline Apple-II Plus central processor does not vary appreciably from installation to installation, although enterprising companies have recently marketed accessories which can actually replace the 6502 microprocessor with the more popular Z80 CPU (and/or increase the clock

speed). The possible systems, however, and the languages available on those systems are too numerous to mention. One special purpose and six general purpose "slots" are available at the rear of the computer for connection of peripheral devices. These slots are supplied power and have access to all data bus, address bus, control and interrupt lines.

The table below describes the configuration of the system used in this investigation.

Table 4: Configuration of the Apple-II Plus Personal Computer Installation

Hardware:

No. used	Device	Remarks
1	Apple-II Plus	48K RAM
1	Apple Language System	16K RAM
2	Apple Disk-II Interface Card	
4	Apple Disk-II Floppy Disk Drives	
1	Apple Parallel Printer Interface Card	
1	Microline-80 Dot Matrix Graphics Printer	
1	Sony Trinitron Color Video Monitor	

Software:

Apple FORTRAN
Apple Pascal

Slot Connections:

Slot No.	Device	Remarks
0	Language System Card	
1	Printer Interface Card	
5	Disk-II Interface Card	Volumes 11 and 12
6	Disk-II Interface Card	Volumes 4 and 5

b. Machine Precision

The Apple-II Plus stores floating point numbers between $1.7 \times 10^{+38}$ and 5.8×10^{-38} in absolute value and

integer numbers between +32,768 and -32,768 [8]. Floating point variables each occupy 4 bytes (two machine words) of storage and integer variables occupy 2 bytes (one word). The bit allocation for a floating point number is:

Bit:	31	30 . . . 23	22 . . . 16 and 15 . . . 0
Item:	sign	exponent	mantissa

Floating point variables having a maximum of 6 decimal digits may be stored.

Using the methods of chapter 3 of reference [7], the estimate of worst case roundoff error is:

$$10^{-6+1} = 0.00001$$

The machine epsilon (smallest floating point number distinguishable from zero) was investigated using the program of section 2.2 of reference [7] and was found to be .119209E-06.

c. The Apple FORTRAN Language and the Pseudo Machine

References [9] and [10] discuss the Apple FORTRAN[3] language and explain its implementation on the machine and relation to the Apple Pascal Operating System. Although it is not necessary to know any Pascal to operate the system and use Apple FORTRAN, from a user point of view it is helpful to remember that this version of FORTRAN was implemented

---

[3] Apple FORTRAN is a registered trademark of the Apple Computer Company.

exclusively for use with the operating system. When using the Apple-II described in this thesis, the user is really working on a virtual or "pseudo" machine which has been created by the operating system. This is best thought of as a software generated device which emulates the familiar hardware (e.g., registers) found in a real computer.

The machine language of this virtual microcomputer is "pseudo code" (p-code) which resembles a true machine language. P-code is, supposedly, portable among all computers which operate under any version of University California, San Diego, Pascal (UCSD Pascal).[4] This would also require that the mass storage medium upon which the program is stored and the format of storage be compatible with both machines (e.g. an Apple-II 5 inch floppy disk is not a compatible storage medium with the IBM 3033 at the Naval Postgraduate School). A p-code interpreter program, which is a true machine language program and is therefore computer dependent converts the p-code to (in the Apple-II case) 6502 instruction codes. All other programs, including the operating system itself, are in p-code.

---

[4]UCSD Pascal is a registered trademark of the Regents of the University of California. Use thereof in conjunction with any goods or services is authorized by specific licence only and is an indication that the associated product or service has met quality assurance standards prescribed by the University. Any unauthorized use thereof is contrary to the laws of the State of California.

It is clear then how the FORTRAN Language option works on the Apple-II Plus. The Apple FORTRAN Compiler is a p-code program which reads TEXT files having proper FORTRAN syntax and in turn creates from them a pseudo CODE file which is compatible with the virtual machine. It is also clear that Pascal and FORTRAN routines can be mixed, through linking, into a single executable CODE file. One of the original advantages of this microcomputer system was that FORTRAN was available and that the great volume of scientific program which are currently in the literature were potentially adaptable to the Apple-II. The author considered that the need to use Pascal would nullify this advantage. Unless the reader desires to use the Pascal unit presented in Appendix A of this thesis, the Pascal language need not be mentioned again.

The "proper" FORTRAN syntax mentioned above is described in detail in Reference [9]. Apple FORTRAN is based on the American National Standard Programming Language FORTRAN ANSI X3.9-1978[5] (popularly known as ANSI subset FORTRAN 77). There are certain deviations from the standard, both deficiencies and extensions, which are adequately discussed in [9].

---

[5]Available from the American National Standards Institute, Inc., 1430 Broadway, New York, New York, 10018.

Readers familiar with FORTRAN will have no trouble following the programs given in this thesis. Some specialized statements, called compiler directives, will be unfamiliar to most users but are discussed in detail in Appendix A. Input/Output statements will briefly be discussed in the next section.

Double precision calculations are not possible in Apple FORTRAN as it is presently implemented.

d. Disk Mass Storage Considerations

Physical records on the Apple-II diskette are called blocks and each consist of 512 bytes. Page 25 of Reference [10] contains a discussion of the technical details of the diskette and its storage format and is summarized in the table below:

Table 5: Selected Characteristics of the Apple Disk-II Flexible Disk Drive

Storage Capacity:	
Bytes (total)	143,360
Tracks	35
Sectors	16
Physical records/blocks (maximum)	280
Physical records/blocks (usable)	274
Files (Directory size limit)	77

Disk files on the Apple-II may be any combination of sequential or direct access and formatted or unformatted. The OPEN statement in Apple FORTRAN corresponds to the more familiar DEFINE FILE statement. The format of the OPEN statement is:

```
      OPEN (u,FILE=fname [,ACCESS='string1']
     + [,STATUS='string2'] [,FORM='string3']
     + [,recl=r1])
```

,where,

OPEN (u

is required and must appear as the first argument and "u" may be an integer variable or expression.

,FILE=fname

is required and must appear as the second argument and fname is a CHARACTER expression of the complete file name (including disk, name and type extension).

,STATUS='string1'

is optional and 'string1' must be one of the CHARACTER constants 'OLD' or 'NEW' (the default is 'OLD').

,ACCESS='string2'

is optional and 'string2' must be one of the CHARACTER constants 'SEQUENTIAL' or 'DIRECT' (the default is 'SEQUENTIAL').

,FORM='string3'

is optional and 'string3' must be either the CHARACTER constant 'FORMATTED' or 'UNFORMATTED' (the default is 'FORMATTED').

,RECL=r1

required and only allowed for direct access files where "r1" is an integer variable or expression.

One serious limitation is present when using direct access files in Apple FORTRAN with the Pascal Operating System. When locating data in the direct access diskette file, the system apparently uses the record number and record length given by the user to compute the number of the bytes from the beginning of the file to the data to be read or written. Since the largest integer which may be represented in the machine is 32,768, this is the maximum number of bytes which may be present in a direct access file. For single direct access files the maximum number of records is (32768 / RECL), so that for the optimum record length of 512 bytes, a direct access file may be at most 64 blocks long.

II. COMPARATIVE SYSTEM TESTING USING OUT OF CORE LINEAR EQUATION SOVLERS

The key factor in determining the suitability of a particular microcomputer system for solving finite element problems is the machines facility for solving systems of simultaneous, linear, algebraic equations. It was decided that, before proceeding with the installation of a code on either chosen microcomputer, a standard test should be devised to ensure that reasonalby large systems of equations could be solved. As a point of interest, it was also desired to compare the relative speeds of the two machines, although neither would be excluded from the study solely because of low solution speeds. Since microcomputers are so much less expensive than larger computers, the relative cost of CPU time is also proportionately much lower, and a slow machine which is sufficiently accurate and has the capability of handling a program large enough for a variety of element types, remains acceptable. Before discussing the choice of solution methods used in the testing, it is helpful to recall the source and characteristics of the systems of equations to be solved.

A familiar and most common method of finite element analysis is the displacement based method (see, for example, [12]) which gives rise to a system of constant coefficient differential equations of the form:

$$M\ddot{U} + C\dot{U} + KU = F$$

,where,

M is the constant coefficient, mass matrix of the finite element structure and usually considers the element mass as a consistent, equivalent masses lumped at appropriate nodes.

C is the constant coefficient, damping matrix of the finite element structure. The coefficients of this matrix are frequency dependent and are best determined from the assembled mass and stiffness matrices and consideration of experimental results [12].

K is the constant coefficient, stiffness matrix of the finite element structure. This matrix is, in the direct stiffness method, assembled from element stiffness matrices, which in turn are developed from strength of materials considerations and the geometry of the chosen element type.

U is a matrix of time dependent, global, nodal displacements in the degrees of freedom appropriate for the element type chosen and the physical structure that the element assemblage represents. The coefficients in $\dot{U}$ and $\ddot{U}$ are understood to be the global, nodal velocities and accelerations, respectively.

F is a matrix of consistent, applied forces acting in the global degrees of freedom and arising from body, surface and concentrated forces applied to the element structure and from initial stresses (e.g. thermal or "lack-of-fit" pre-stresses). In the most general case, these forces will also be functions of time.

For static analysis M and C effects are disregarded and:

$$K\ U = F$$

,where, the entries of U and F are independent of time and the system of differential equations reduces to a system of linear algebraic equations. An over simplified, but helpful way to understand this system is to recall a simple Newtonian summation of forces, including D'Alembert's dynamic forces, on a single mass (i.e., F = ma); the two situations are analogous, the finite element system being more general and most powerful. Even when dynamic effects are considered, direct integration methods [12] are most often used and require only the solution of the general form at various instances in time, such that one is again solving a linearized algebraic system. The equations produced by the above method have several desirable properties [12-14].

Firstly, the system of equations is generally well conditioned and positive definite [14]. For this class of

problem Gaussian elimination or equivalent techniques (e.g. see LDL^T factorization in [12]) are the most efficient and stable [13].

Secondly, the system is symmetric, such that the matrix of coefficients has the appearance of having been reflected about the main diagonal. In the interests of reducing the number of arithmetic operations and the storage required for solution, only the main diagonal and all entries above (or below) need be considered. This is the most easily programmed improvement over full Gaussian elimination and requires the least amount of additional code [13].

Thirdly, the coefficient matrices are sparse and tightly banded, if proper techniques of nodes numbering are used [12, 14]. Sparsity means there are a large number of zeros in the off-diagonal entries of the matrix and bandedness means that most non-zero entries are concentrated near the main diagonal. With some additional code, the solution algorithm can be designed to store and work only with elements "below the skyline" of the matrix [12], and to have some reduction in storage requirements and solution times. Note, however, that for smaller systems of equations and for microcomputers, this reduction in storage is somewhat offset by the additional code required. With an interpretive language such as BASIC, some of the speed advantage is also offset when this additional code must be repeatedly read.

Since the exactness of the finite element solution depends in the large part on the fineness of the mesh (greatest possible number of discrete elements to model the continuous real system), it is not uncommon for a practical problem to have from hundreds to tens-of-thousands of elements [12]. No computer presently in use can solve the largest problems by having the entire system of equations in core throughout the entire solution phase. Most new, general purpose codes work with out-of-core solution of equations and store the majority of the system on magnetic disk or tape as part of a more complete problem database. Any useful algorithm must be capable of solving large numbers of equations, efficiently, out-of-core.

A. CHOICE OF EQUATION STORAGE SCHEME AND SOLUTION ALGORITHM

For the purposes of the equation solution testing phase of this thesis, the algorithm of Reference [13] was selected. This algorithm was perceived to be well suited to the general test role for three reasons. Firstly, the number of equations which may be solved is theoretically limited only by the amount of out-of-core storage available to the computer. Secondly, the amount of core memory required for solution is uncoupled from the number of equations and may be independently selected when choosing the "block" size (the size of the square submatrices into which the complete systems is divided). No matter how large a percentage of the microcomputer memory is

taken up by the solution program, a sufficient (if not optimum) amount of core storage should remain for an acceptable block-size to be chosen. Lastly, the code is very simple to follow and requires the fewest number of instructions of the candidate algorithms. This makes a program easier to debug and more likely to fit into a small machine.

B. THE BLOCK SOLVER METHOD FOR OUT OF CORE SOLUTION OF SYSTEMS OF LINEAR EQUATIONS

The discussion which follows parallels that of Reference [13].

Consider Figure 1, which shows a positive definite, symmetric, sparse, banded matrix in an associated system of linear algebraic equations. The terms represented by "F" may be thought of as loads or generalized forces, such that "U" and "K" might then represent generalized displacements and equivalent stiffnesses, respectively. The general elements "F_N" and "U_N" are themselves vectors having, say, NS components and the general terms "K_{IJ}" are NS-by-NS square sub-matrices. The variable NS is defined as the blocksize, and will eventually determine the amount of core required to solve any arbitrarily large system.

Figure 2 illustrates the block half-bandwidth M (or MM or Mm) and shows that portion of the complete stiffness matrix which must be stored, for this algorithm, to effect a solution. Storage for fully N-by-M blocks of the stiffness matrix is reserved, even though this is more than is

absolutely required to hold one complete half-band. The extra blocks are used for storage of intermediate results of calculations during the solution.

The program requires that an NS-by-NS block of loads, each containing a single portion of the complete load vector, be stored following its associated row of stiffness blocks. In the present form of this algorithm only one "right-hand-side" is stored in each load block and most of the space in these blocks is unused. It would be a simple matter, once the solution procedure is understood, for the reader to modify the program to allow up to NS multiple right hand sides (and only slightly more difficult to allow a greater number).

It is recommended that all blocks (both stiffness and load) be stored in a single, large, one dimensional array, unless the language being used offers special advantages for other storage schemes.

The block solver algorithm itself is simply a form of Gaussian elimination which exploits the favorable characteristics of a system of finite element equations. The reader who is interested in discovering the solution scheme for him or her self is encouraged to complete the following exercise, by hand, with a 4-by-4 example:

1. Construct the first equation by matrix multiplication of block row one (K_{11} - K_{1M}) and the generalized displacements (U_1 - U_N).

$$
\begin{bmatrix}
K_{11} & K_{12} & K_{12} & \cdots & \cdots & K_{1M} & \emptyset & \cdots & \emptyset \\
K_{12}^{T} & K_{22} & K_{23} & K_{24} & \cdots & \cdots & K_{2,M+1} & \emptyset & \cdots \\
K_{13}^{T} & K_{23}^{T} & K_{33} & K_{34} & K_{35} & \cdots & \cdots & K_{3,M+2} & \cdots \\
. & . & . & . & . & . & . & . & . \\
K_{1M}^{T} & K_{2M}^{T} & \cdots & \cdots & K_{M-1,M}^{T} & K_{MM} & K_{M,M+1} & \cdots & K_{M,2M} \\
. & . & . & . & . & . & . & . & . \\
\cdots & K_{I-M+1,I}^{T} & \cdots & \cdots & K_{I-1,I}^{T} & K_{II} & K_{I,I+1} & \cdots & K_{I,I+M-1} \\
. & . & . & . & . & . & . & . & . \\
\emptyset & \cdots & \cdots & \emptyset & K_{N-M+1,N}^{T} & \cdots & \cdots & K_{N-1,N}^{T} & K_{NN}
\end{bmatrix}
\begin{Bmatrix} U_1 \\ U_2 \\ U \\ . \\ U_M \\ . \\ U_I \\ . \\ U_N \end{Bmatrix}
=
\begin{Bmatrix} F_1 \\ F_2 \\ F \\ . \\ F_M \\ . \\ F_I \\ . \\ F_N \end{Bmatrix}
$$

Figure 1. A Hypothetical System of Linear, Algebraic Equations Generated by the Finite Element Method

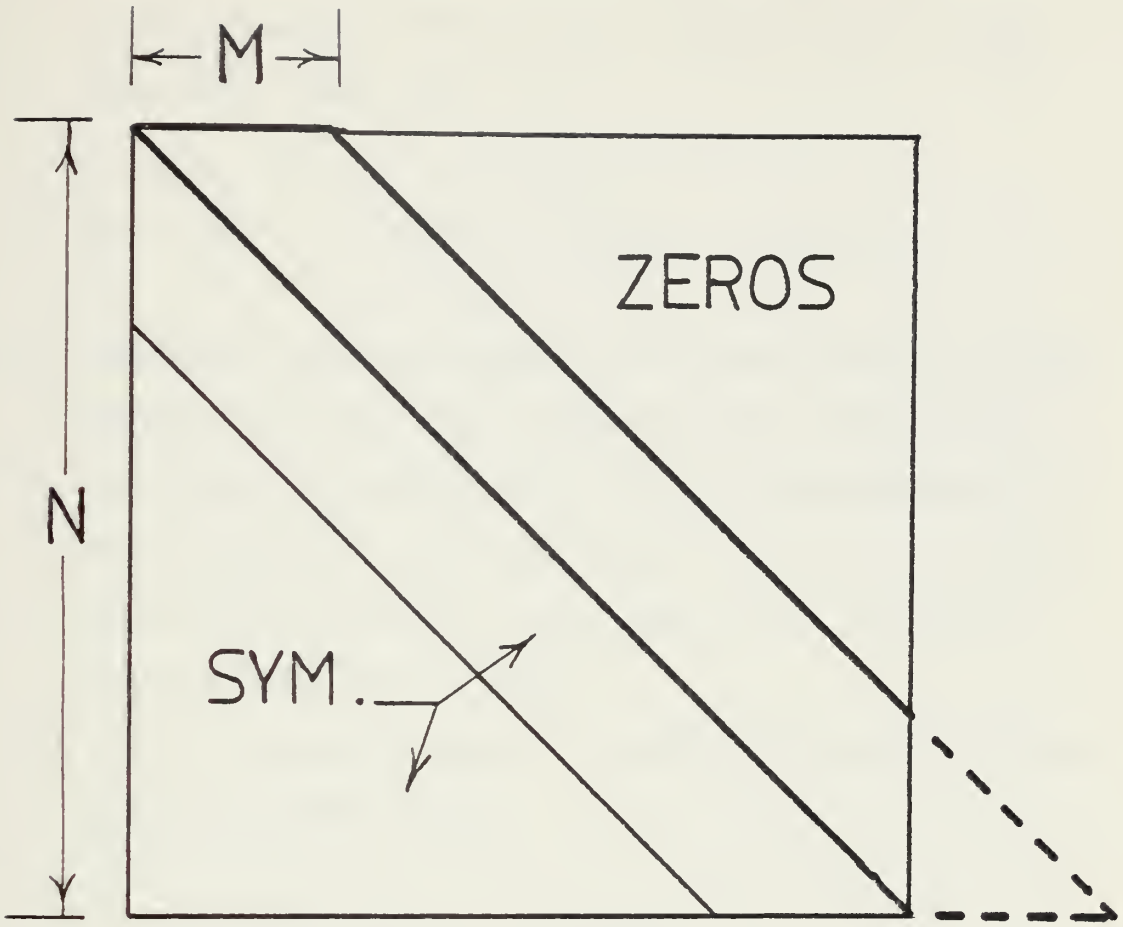

Figure 2. The Banded Character of the Equivalent Stiffness Matrix

2. Solve the first equation for U_1 using the inverse of the diagonal block of the current row, in this case K_{11}^{-1}. That is:

$$U_1 + K_{11}^{-1} \quad (F_1 - K_{12}U_2 - K_{13}U_3 - \ldots - K_{1M}U_M)$$

3. Construct the next equation by matrix multiplication of block row two and the generalized displacements.
4. Substitute the expression for U_1, obtained from the first equation, into the second. Collect the terms which multiply each displacement block and which involve only force blocks.
5. Use the collected terms to reduce load and stiffness blocks as shown below:

$$F_2^* = F_2 \quad - K_{12}^T K_{11}^{-1}$$

$$K_{34}^* = K_{34} \quad - K_{13}^T K_{11}^{-1} K_{14}$$

or, more generally:

$$F_M^* = F_M \quad - K_{1M}^T K_{11}^{-1} F_1$$

$$K_{IJ}^* = K_{IJ} \quad - K_{1I}^T K_{11}^{-1} K_{IJ}$$

6. The expression for U_1 obtained in step 2 must be used to reduce all stiffness coefficients in block rows 2 to M and block columns 2 to M and all load blocks associated with those block rows.
7. Once the reduction in step 6 is complete, the second equation is solved for U_2 and a similar reduction accomplished on block rows and column 3 to M+1 and their associated load blocks.
8. After reduction is complete through the Nth block row, the Nth equation will be:

$$F_N^* = K_{NN}^* U_N$$

This equation is solved directly for U_N, after which back substitution is used to solve all previous equations in the usual manner.

Upon completion of subroutine SOLVE the original stiffness matrix has been destroyed and the displacements reside in the blocks which originally contained the load vector.

The algorithm is coded using five subroutines and one function subprogram (or statement function definition, depending on the language chosen). Subroutine SOLVE is the control subprogram which carries out the steps outlined in the description of the algorithm. Subroutine MULT performs matrix multiplication of two arrays which have been stored

in unidimensional form. Subroutine SYMINV ensures that each diagonal block is truely symmetric, attempts to detect singularity or near singularity by examining the diagonal terms within the diagonal block, calculates and prints a condition number and inverts the block. An appropriate warning message is printed and, if necessary, the program is halted whenever an error condition exists. The condition number used is the product of the maximum column sum within the diagonal block before and after inversion and may be used to estimate the solution accuracy [13]. Subroutines RDDISK and WRDISK are machine dependent subroutines which read a specific record from a random access disk file. The function subprogram used has various names depending on the version (NTRACK, FNTrk,...) but is simply used to find the record number in which a block from block row I and block column J is stored. Other subroutines required to generate test matrices and to retrieve answers and control programs to call all of the above are discussed in the next section.

1. Enhanced BASIC Coding on the HP-9845A

The block solver algorithm was coded for the HP-9845A using subroutines SOLVE, RDDISK, WRDISK, MULT, SYMINV and function DEFinition FNTrk. Since the subroutines are not overlaid, all are required to be in memory at the same time. All transfers within the controlling subroutine (SOLVE) are made to labels rather than statement numbers, so that the

program may be RENumbered at the convenience of the user. In order to decrease the run time of the program REMarks were kept to a minimum, a BUFFER was used and the most frequently called subroutines were placed first in sequence. OVERLAP mode is declared upon entry into subroutine SOLVE to allow simultaneous I/O and computation and SERIAL mode is declared upon exit to avoid disabling error traps in the calling program.

The HP-9845A version of this algorithm is given in Appendix B, along with a test program to generate various size systems of suitable equations (subroutine rest), print the answers (subroutine Answer) and to mimic the FORTRAN EQUIVALENCE statement (subroutine Equiv) which has no analog in BASIC.

A sample problem and program output is provided in Appendix C.

2. Apple FORTRAN Coding on the Apple-II Plus

The block solver algorithm was coded for the Apple-II Plus using subroutines SOLVE, RDDISK, WRDISK, MULT, SYMINV and, within SOLVE, implicit function definition NTRK(I,J). The above subroutines are all compiled into one object/p-code file by imbeding the appropriate $INCLUDE statements at the end of SOLVE. Dummy dimensioning (e.g. AK1(1)) is used in all subroutines and array sizes are determined in the calling program, preferably by some dynamic means using a single workspace.

In order to handle substantial size problems in the face of random access file size limitations, subroutines RDDISK and WRDISK are set up to handle as many as four units (7-10), each of which must have been pre-defined/OPENed in the main calling program. In calling RDDISK or WRDISK, NACTIV is the desired record number, C is the array to be read or written, NENTRY is the one-dimensional size of C and NRECFY is the number of records per random access file.

The Apple-II Plus version of this algorithm is given in Appendix D, along with a series of subroutines and a main program to OPEN the necessary files and calculate file parameters (subroutine FILES), generate various size systems of suitable equations (subroutine TEST), write them into appropriate files (subroutine DISKWR) and read (subroutine DISKRD) and print (subroutine ANSWER) the answers. The control program (program THESIS) is also included in Appendix D and is organized to use OVERLAYS to link all portions of the test program into a single executable CODE file and to manage the workspace. A system of integer pointers (e.g. NAK1, NAK2, ...) divide up a workspace of 3000 floating point variables according to the chosen block size, number of equations and half-bandwidth.

A sample problem and program output is provided in Appendix E.

3. Solution Times

The table below is a summary of all the test runs made on each machine. Where the name of a system is absent, that particular combination of parameters, for one reason or another, was not attempted (e.g. the blocksize of 2 was not tested on the Apple-II because the Disk Drive was already continuously busy with a block size of 4).

For comparison purposes, the solution times obtained on a TEKTRONIX 4081 Graphics System Mini-computer with hard disk drive are included in the table. For more information on the TEKTRONIX version of the block solver, the reader is referred to Reference [15].

Sample solutions of systems of 160 equations with true half-bandwidths of 64 appear in Appendices C and E.

Table 6: Solution Times for Standard Equation Solving Trials

Number Block Rows (NN)	Block Half-Bandwidth (MM)	Square Block Size (NS)	Record Length In Bytes	Computer System Used	Solution Time (hr:min:sec)
16	16	2	32	HP9845A	0:7:45.44
8	8	4	128	HP9845A	0:2:23.57
			64	Apple-II	0:7:45.13
4	4	8	512	HP9845A	0:3:34.93
			256	Apple-II	0:5:28.67
2	2	16	2048	HP9845A	0:1:53.10
			1024	Apple-II	0:4:51.89
1	1	32	8192	HP9845A	0:2:41.67
			4096	Apple-II	0:5:29.93
4	2	8	512	HP9845A	0:0:52.33
			256	Apple-II	0:2:50.04

Table 6 (contd)

Number Block Rows (NN)	Block Half-Bandwidth (MM)	Square Block Size (NS)	Record Length In Bytes	Computer System Used	Solution Time (hr:min:sec)
6	3	8	512	HP9845A	0:2:22.92
			256	Apple-II	0:7:55.01
				TEK4081	0:0:09
8	4	8	512	HP9845A	0:5:05.99
			256	Apple-II	0:16:20.74
				TEK4081	0:0:23
12	5	8	512	HP9845A	0:11:48.64
			256	Apple-II	0:37:03.50
				TEK4081	0:0:40
16	6	8	512	HP9845A	0:22:15.91
			256	Apple-II	1:08:37.68
				TEK4081	0:1:17
20	7	8	512	HP9845A	0:37:14.14
			256	Apple-II	1:51:31.04
				TEK4081	0:2:10
20	8	8	512	HP9845A	0:45:47.61
			256	Apple-II	2:13:35.89
				TEK4081	0:2:28

C. COMPARISON AND DISCUSSION OF RESULTS

Problem solution times for both the HP-9845A and Apple-II Plus greatly exceeded those of the TEKTRONIX 4081 for identical systems of equations. Note that the TEKTRONIX 4081 never needed 2 1/2 minutes for any problem attempted while both of the test systems used more than this on most square sets of 32 equations. In general the HP-9845A needed only one third the solution time required by the Apple-II Plus.

Although it was originally intended that the TEKTRONIX 4081 single precision solution should be the standard for accuracy (i.e., the "exact" answer), it was discovered that the number of significant digits routinely carried in

HP-9845A computations exceeded that of the TEKTRONIX. This more precise machine word resulted in the answers of the HP-9845A being more accurate than those of the TEKTRONIX 4081 double precision solution. Using the HP-9845A answers as the true solution, the Apple-II Plus was found to be accurate to three significant figures. The more than two hours of computation (including 20 matrix inversions) required of the Apple-II Plus for the largest problem was considered a sufficiently rigorous test of machine accuracy.

The first five runs shown in Table 6 were all square systems of 32 linear algebraic equations in 32 unknowns. The major difference in these problems was the record length, as determined by the square blocksize and floating point word size, although the problems with larger blocksizes were progressively more dense. This density difference was negligible for the algorithm chosen because the skyline within a given block is never examined nor used to reduce the number of arithmetic operations. It was expected that optimum record lengths would be those which matched the physical record size on the respective disk drive system and so result in the fastest Input/Output performance. In some cases, for the chosen algorithm, this particular size was not achievable (i.e., when the disk physical record length in bytes divided by the floating point word size in bytes was not a perfect square). The optimum record length is also affected by the relative

amounts of I/O and computation in a program and the ability of a machine to handle both simultaneously. The observed optimum block size for the HP-9845A and Apple-II Plus was 16 X 16. The remaining test combinations were run to gauge the amount of time to be expected for the solution of larger systems and to search for machine peculiarities which might place a practical limit on the number of equations.

III. IMPLEMENTATION OF REPRESENTATIVE CODES ON THE TEST SYSTEMS

Upon completion of initial testing of the equation solving capacity of the HP-9845A and Apple-II Plus a decision was made to continue the study by implementing a finite element code on these systems. The plan was to adapt an existing code to solve for forces and displacements in space trusses with static loading and to which other element and problem types could later be added. This adapted code would be re-programmed and modified to run on the test machines and be exercised on a number of problems of suitable difficulty. It was considered mandatory that the source program(s) use an out-of-core equation solver and desirable that some automatic node and element generation features be provided, with input data in a format familiar to users of general codes. Toward this end, a search was undertaken to find an appropriate source program.

A. PREVIOUS WORK ON THE HP-9845A

Reference [3] describes Mallory's success in modifying STAP [12] to produce SSAP-NPS for the HP-9845A with tape mass storage. SSAP-NPS solves 2-D and 3-D trusses using an in-core equation solver. Although some thought was given to the possibility of modifying it to solve out-of-core and to adding additional elements, time constraints finally

dictated that this idea be abandoned in favor of developing a code for the Apple-II Plus. STAP-NPS was, however, modified by changing the mass storage unit specifiers in the program to select the newer flexible disk drives rather than the older tape units. Test runs using Mallory's original problems were made with significant reductions in solution time.

B. APPLE-II PLUS IMPLEMENTATION OF STAP-NPS

The challenge of implementing a finite element code on the Apple-II Plus really involved overcoming the limitation on the number of subroutine/function calls which may be made by a program (see Appendix A). The random access file size limit (32768 bytes maximum length) was not reached in any of the test problems run in for the thesis (STAP-NPS will presently not allow the stiffness matrix to occupy multiple files). The slow speed and three significant figure accuracy observed in the Apple-II were already considered in the first phase of the study.

1. Software Sources

Many of the popular codes in use today are extensions of the pioneering work of Dr. Edward L. Wilson of the University of California at Berkeley. It was from such sources that STAP-NPS evolved.

The main program, exclusive of element dependent subroutines, was taken from the program STAMOD[6] by Nor, while truss element subroutines were taken from Reference [12].

2. Program Development

The preliminary development of STAP-NPS for the Apple-II was done using an IBM 3033 mainframe computer and IBM FORTRAN at the W. R. Church Computer Center, Naval Postgraduate School, Monterey, California. STAMOD was loaded from an existing card deck while subroutines RUSS and TRUSS [12] were entered by hand. Missmatch problems in COMMON blocks and SUBROUTINE calls were resolved and the database structure was set up. The resulting program was then converted from double to single precision (the Apple-II has only single precision) and tested on several problems.

Up to this point STAP-NPS was one single program. The next step was to split the program up into separate segments, each of which had the maximum allowable number and depth of FUNCTION and SUBROUTINE calls and which would fit in the Apple-II's memory. Estimates of what would "fit" were based on previous experience and careful study of the number of variables needed in memory for each major program phase. It was finally decided to break the program into seven parts.

---

[6] An unpublished program used in supporting studies for "REALISATION ET ETUDE D'ELEMENTS DE COQUE DE MINDLIN," Sopha Nor, Doctoral Dissertation, Universite de Technologie de Compiegne, France, 29 June 1978.

3. STAP-NPS

The FORTRAN names of the seven PROGRAM segments are PROBLM, ELEMS, LOAD, BLOCKS, ASEMBL, SOLVE and STRES. Note, however, that these programs will identify themselves by a more complete title when interactively communicating with the user (e.g., ASSEMBLE vice ASEMBL for the fifth segment). Each segment has certain major functions which it is expected to accomplish and certain subroutines with which it is associated.

Program PROBLM determines the language in which the output is to be printed (both French and English are available), reads control parameters and nodal point information, automatically generates nodes as directed, reads and stores boundary conditions for global, nodal degrees of freedom and determines the total number of equations to be solved.

Program ELEMS calls the appropriate element subroutin (only a truss element is currently available), reads and stores connectivity information, automatically generates elements as directed and calculates the characteristics of the global stiffness matrix.

Program LOAD reads in loading information and calculates and stores load vectors.

Program BLOCK considers problem size and user recommended block size and determines the actual blcoksize to be used. Direct access record length and actual block size are then used to develop an indexing scheme for locating any

coefficient in the global stiffness matrix within its random access file. The number of columns of the compact, global stiffness matrix and the first coupled blocks, per block, are calculated and stored.

Program ASEMBL calls the appropriate element subroutine and calculates and stores the compact form of the global stiffness matrix.

Program SOLVE has two separate modes of operation. The first time SOLVE is run LDL^T factorization [12] is performed on the stiffness matrix, the first loadcase vector is reduced and multiplied by the resulting factored form, back substitution is carried out and the answers (i.e., the nodal displacements) are stored for calculation of stresses. In subsequent calls successive loadcase vectors are retrieved and used to calculate their respective displacements.

Program STRES calls the element subroutine, prints the nodal displacements and calculates and prints the element stresses.

a. Major Variables

The major variables used by STAP-NPS are shown below:

NUMNP	Number of nodal points.
NUMEG	Number of element groups.
NUMMAT	Number of materials.
NUME	Number of elements.

NDOF	Global degrees of freedom possible per node.
NEQ	Number of equations.
NLOAD	Number of loads within a given loading case.
NBLOCK	Number of blocks into which the global stiffness matrix is divided for solution purposes.
ISTOTE	Desired number of coefficients per block.
ISTOH	Actual number of coefficients per block as calculated by the program based upon the remaining workspace and the problem size.
X(NUMNP)	X coordinate of each nodal point.
Y(NUMNP)	Y coordinate of each nodal point.
Z(NUMNP)	Z coordinate of each nodal point.
ID(NDOF*NUMNP)	Contains a one for each inactive degree of freedom and a zero for each active degree of freedom. Zeros are later changed to the equation number which will provide the solution for that D.O.F. and node and the ones are changed to zeros [12, 19].
E(NUMMAT)	Young's modulus for each material group used in the structure.
AREA(NUMMAT)	X-sectional area for each material group used in the structure.
XYZ(6 * NUME)	Each column contains the cartesian coordinates which apply to a single element. Rows 1-3 and 4-6 contain information on the "i" and "j" ends of the element, respectively.

MHT(NEQ)	Vector of column heights in the global stiffness matrix. This information allows a compacted (having fewer zeros) one-dimensional form of the global stiffness matrix to be stored and accessed. The skyline may be reconstructed from the MHT vector [12].
MAXA(NEQ + 1)	Stores the addresses of the diagonal elements of the compacted global stiffness matrix. MAXA(1) is always one and MAXA(NEQ + 1) is always the total number of elements in the compact stiffness matrix plus one.
LM(6 * NUME)	Each column of this array holds connectivity information for one element. Each row represents a local element degree of freedom and each entry in LM is a global equation number taken from the ID array.
MATP(NUME)	Contains the material type of each element. The entries in this vector become NUMMAT for retrieving the Young's modulus and x-sectional area.
R(NEQ)/V(NEQ)	A single load vector describing one loadcase (i.e., a "right-hand-side"). R is calculated from the individual global, nodal loads in FLOAD and the ID array.

FLOAD(NLOAD) Vector of individual, concentrated forces applied to global nodes within one loadcase.

NOD(NLOAD) Vector of global node numbers to which concentrated forces are applied within one loadcase.

IDIRN(NLOAD) Vector of principal cartesian coordinate directions along which nodal forces are applied within one loadcase.

NCOLBV(NBLOCK) Vector containing the number of columns stored within a given block of the compressed stiffness matrix.

ICOPL(NBLOCK) Vector containing the number of the first block which is coupled to another particular block of the compressed stiffness matrix.

AA or A or B (ISTOH) Various blocks of the compressed stiffness matrix during the assembly, factorization and back substitution phases.

D(NEQ) Vector of nodal displacements in the global degrees of freedom (i.e., the answer to the problem).

b. Management of Variable Storage Space

In programming a microcomputer it is convenient to use one single, large array called a "workspace" for storing most variables which will be READ or calculated by the program. The size of the one large array may be adjusted to allow the maximum number of variables to be present in

memory along with the program instructions. Pointers are used to keep track of and locate sub-arrays within the workspace and are used to pass the locations in SUBROUTINE or function CALLs. Because the Apple-II does not use the same number of bytes for the storage and representation of integer numbers and floating point numbers, two separate workspaces (one integer and one real) must be used. Normal schemes of incrementing the pointers differently to locate intermingled floating point and integer sub-arrays cannot be used.

The integer workspace is the array IA which is dimensioned MTOTI and is contained in COMMON block IWORK. The real workspace is the array A which is dimensioned MTOTR and is contained in COMMON block RWORK. Integer and real pointers begin with NI and NR, respectively, and change their true position within the workspaces, during each major phase of the problem, as the variables they represent are shuffled about. When the workspaces are passed to subroutine RWCOMN the pointers NENDI and NENDR mark the highest numbered elements of IA and A which must be written into and, in the next segment, read from mass storage. The following two tables show the pointer allocations throughout the solution process:

Table 7: Integer Workspace Pointer Allocation

Segment:	PROBLEM	ELEMENTS	LOADS	BLOCKS	ASSEMBLE	SOLVE	STRESS
Integer Pointer							
NI1	ID	ID	ID	MHT	MHT	MAXA	MAXA
NI2	not used	MHT	MHT	MAXA	MAXA	NCOLBV	NCOLBV
NI3	not used	MAXA	MAXA	LM	NCOLBV	ICOPL	ICOPL
NI4	not used	LM	LM	MATP	ICOPL	ID	ID
NI5	not used	MATP	MATP	NCOLBV	LM	LM	LM
NI6	not used	not used	NOD	ICOPL	MATP	MATP	MATP
NI7	not used	not used	IDIRN	not used	not used	not used	not used

Table 8: Real Number Workspace Pointer Allocation

Segment:	PROBLEM	ELEMENTS	LOADS	BLOCKS	ASSEMBLE	SOLVE	STRESS
Real Pointer							
NR1	X	X	E	E	AA	A	A
NR2	Y	Y	AREA	AREA	E	B	B
NR3	Z	Z	XYZ	XYZ	AREA	D	D
NR4	not used	E	R	R	XYZ	V	V
NR5	not used	AREA	FLOAD	not used	not used	E	E
NR6	not used	XYZ	not used	not used	not used	AREA	AREA
NR7	not used	not used	not used	not used	not used	not used	XYZ

The pointers are modified in several programs and subroutines in STAP-NPS. Each time the numerical value of the highest active pointer is increased a check must be made to ensure that no element of the workspace will be stored outside its dimensioned area. The magnitude of the topmost active pointer is compared to MTOTI or MTOTR, as appropriate, and if the required value is too large subroutine ERROR is called. Subroutine ERROR prints a diagnostic message showing the amount by which the offending pointer exceeds the memory available and the problem phase which cannot be accomplished. The user then has the option of attempting to set MTOTI and MTOTR closer to the machine limits, rearranging the mesh into smaller pieces (e.g., by using multiple element groups) or abandoning the Apple-II for a larger machine and program.

The first six segments of STAP-NPS all have entry and exit diagnostics which will display the contents of the integer and floating point workspaces. The printing of these diagnostics may be declined by the user. They were first included in the programs to assist in debugging and were allowed to remain so that users could more easily modify or add to the code. These diagnostics could also be of value to the beginning student of the finite element method.

c. The Database and Communication Between Program Segments

Communication between programs was accomplished by including a new subroutine (RWCOMN) in each segment. This subroutine writes the contents of the entire COMMON block (which is the same group of variables in each segment) and all other necessary integer and real variables into an unformatted, serial disk file. All segments, except the first, resume solving the problem after calling RWCOMN with an argument of "1", thereby receiving the necessary data from previous calculations and READs. All segments call RWCOMN with an argument of "2" as their final action before closing file ICOM and stopping. The old ICOM file from the previous segment is deleted just before the new one is written, well after any run time errors could affect its contents.

Program STAMOD was already organized around six FORTRAN I/O units which naturally evolved into the remaining database files for STAP-NPS. These database files are:

IIN

This sequential, formatted file contains the input data for the problem to be solved. IIN is an internal name only; the actual name of this file is entered interactively by the user. (If CONSOLE:$ is entered then IIN becomes the keyboard.) It is created by the user with the Apple E(ditor before program PROBLM is run and is read by programs PROBLM, ELEMS and LOAD.

IOUT

This sequential, formatted file contains all printed output for the problem to be solved. IOUT is an internal name only; the actual name of this file is entered interactively by the user. (If a file other than PRINTER:$ or CONSOLE:$ is specified, the contents must be examined before the next segment is run or be lost.)

IDTAP

This sequential, unformatted file contains the ID array. It is created in program PROBLM and read in program SOLVE.

IELMNT

This sequential, unformatted file contains an indexing parameter (MIDEST), the element type (NPAR1) and NUME, NUMMAT, E, AREA, XYZ, LM, and MATP for each element group. It is created in program ELEMS and read in programs ASEMBL, SOLVE and STRESS.

ILOAD

This sequential, unformatted file contains R vectors from the various load cases. It is created in program LOAD and read in program SOLVE.

IRIG

This random access, unformatted file contains the condensed, and later factored, form of the global stiffness matrix. It is created in program ASEMBL and read in program SOLVE. The record length is user selectable by changing the variable LONG in program PROBLEM.

ICOM

This sequential, unformatted file contains the active portions of the real and integer workspaces and the contents of all other COMMON blocks. It is created or re-written in each program segment and all but the first read this file through subroutine RWCOMN.

4. Running STAP-NPS

In order to "X(ecute" the programs in STAP-NPS the following disks, at a minimum, are needed:

FORT1:

Contains APPLE FORTRAN (Pascal) Operating system. Also contains the F(iler and E(ditor for creating input data files for the problem to be solved.

DATA1: (or DATA3:)

Contains executable CODE files for the first five segments/programs in STAP-NPS. Also contains the input data TEXT files for the four problems used for original program testing.

DATA2: (or DATA4:)

Contains executable CODE files for the last two segments/programs in STAP-NPS.

TWO SCRATCH DISKS

Any two formatted disks having sufficient space for database files. These two disks must NOT be write protected. Normal program execution will not damage pre-existing files.

To actually run the programs:

(1) Load volume 4 with FORT1:, volume 5 with DATA1: and volumes 11 and 12 with scratch diskettes.

(2) Turn on the monitor and other peripherals. Energize the Apple-II Plus and observe initial program loading of the FORTRAN Operating System.

(3) Enter the F(iler and set the date. Enter the E(ditor and create file IIN using the format shown in the following subsection.

(4) X(ecute programs #5:ONE.CODE through #5:FIVE.CODE. These are simply the executable p-code files of programs PROBLM through ASEMBL. Interact and provide information as requested.

(5) Remove DATA1: from volume 5 and replace it with DATA2:

(6) X(ecute programs #5:SIX.CODE and #5:SEVN.CODE (executable p-code files of SOLVE and STRESS) alternately until all loadcases have been solved and the displacements and stresses printed.

a. Input Data Card Formats

Input data cards are actually lines of data in database file IIN. The user is required to construct this file using the E(ditor, prior to running STAP-NPS.

Language Option Card (20A4)

Note	Columns	Variable	Entry
(1)	1-4	Talk	FRAN for French ENGL for English

Notes:

(1) This card must be included. Default language is English.

Heading Card (20A4)

Note	Columns	Variable	Entry
(1)	1-80	HED	Problem title

Notes:

(1) This card must be included.

Master Control Card (1I5,6I1,1I4,3I5)

Note	Columns	Variable	Entry
	1-5	NUMNP	Total number of nodal points
(1)	6-11	IDOF	EQ. 0 free EQ. 1 fixed
	12-15	NUMEG	Number of element groups
	16-20	NLCASE	Number of load cases
	21-25	MODEX	EQ. 0 Data check EQ. 1 Solution
	26-30	ISTOTE	Desired number of coefficients per block

Notes:

(1) For two dimensional trusses enter "001111."
For three dimensional trusses enter "000111."
Default is "000000" which will use excessive diskette storage and memory for the solution.

Nodal Point Definition Cards (A1,I4,A1,I4,5I5,3F10.0,I5)

Note	Columns	Variable	Entry
	1	IT	Blank for X,Y,Z, EQ. X Cylindrical
(1)	2-5	N	Node number
(2)	6	JPR	Print control see Note (2)
(3)	7-10	IDT(1)	EQ. 0 free EQ. 1 fixed (EQ.-1 fixed for automatic gener.)
(3)	11-35	IDT(2)-IDT(6)	EQ. 0 free EQ. 1 fixed (EQ.-1 fixed for automatic gener.)
	35-45	X	X coordinate of node N
(4)	46-55	Y	Y coordinate or radius - node N
(4)	56-65	z	Z coordinate or theta - node N

Note	Columns	Variable	Entry
(5)	66-70	KN	Node increment for automatic generation

Notes:

(1) Need not be input in numerical order but eventually must read in nodes 1 through NUMNP. Last node read in must be node NUMNP.

(2) Print control character only read from card for node number 1. Codes are:

Blank - no print supression

EQ. A - supress list of node coordinates

EQ. B - supress list of equation numbers

EQ. C - supress both of the above

(3) These are local degrees of freedom and should be fixed where global conditions so require. When two consecutive cards are being used to automatically generate elements, all fixed degrees of freedom must be set to "-1" vice "1".

(4) Cylindrical coordinates only if IT is non-blank.

(5) Nodes generated in the sequence:
NODE1,NODE1+(1*KN1),NODE1+(2*KN1),. . ., NODE2
where NODE1 is N on the first, KN1 is KN on the first and NODE2 is N on the second of two consecutive Nodal Point Definition Cards. Default value of KN is "1".

Element Data Block Marker Card (1A4)

Note	Columns	Variable	Entry
(1)	1-4	BLOCK	ELEM

Notes:

(1) This card must be included. It is the marker for Program ELEMS to begin reading element data from file IIN after unit rewind.

Material Group Specification Cards (I5,2F10.0)

Note	Columns	Variable	Entry
	1-5	N	Material group
	6-15	E	Young's Modulus
	16-25	AREA	Truss element x-sectional area

Notes:

None.

Element Data Cards (5I5)

Note	Columns	Variable	Entry
(1)	1-5	M	Element number
	6-10	II	Node number at one end
	11-15	JJ	Node number at other end
	16-20	MTYP	Material type
(2)	21-25	KG	Increment for automatic element generation

Notes:

(1) Put in ascending order beginning with number "1".

(2) Missing elements are filled in by using the material type of the last card before the missed element and incrementing the element numbers by one and the node numbers by KG until the gap is closed.

Loading Data Block Marker Card (1A4)

Note	Columns	Variable	Entry
(1)	1-4	BLOCK	LOAD

Notes:

(1) This card must be included. It is the marker for Program LOAD to begin reading loading case data from file IIN after unit rewind.

Load Case Control Cards (2I5)

Note	Columns	Variable	Entry
(1)	1-5	LL	Loading case number
	6-10	NLOAD	Number of loads within the load

Notes:

(1) Enter is ascending order beginning with "1".

Concentrated Load Cards (2I5,F10.0)

Note	Columns	Variable	Entry
	1-5	NOD	Node to which load is applied
(1)	6-10	IDIRN	Pinciple direction
	11-20	FLOAD	Load magnitude

Notes:

(1) Positive directions only:

EQ. 1 - +X

EQ. 2 - +Y

EQ. 3 - +Z

For negative directions change the sign of FLOAD. Language Option Cards, Heading Cards, Master Control Cards and Data Block Marker Cards will appear only once in every file. Other cards will appear some number of times, dependent upon the particular problem being solved.

C. TESTING OF STAP-NPS

STAP-NPS was tested on four problems, the formatted input files of which appear in Appendix G. Problems were chosen for their varying degree of difficulty, the combination of features which they tested and the availability of a previous, reliable solution.

Truss number one was a simple three member problem, easily solvable by hand, which was used to debug the program. Truss number two had more nodes and elements and tested the multiple load case feature. Truss number three was a large crane with multiple materials and automatic node generation. Truss number four was a problem taken from Mallory's thesis [3] and tested the automatic element generation feature of the program and three dimensional operation. The cylindrical coordinate option and print supression features have never been tested.

The English language program output from solving truss number four is given in Appendix H. The output of the first segment for the same problem with French as the chosen language has also been included.

IV. CONCLUSIONS

In concluding, two separate questions will be addressed. Firstly, what general techniques should be used when tailoring a finite element code to a microcomputer? Secondly, how well suited to finite element work were the two test systems and what limitations were encountered in each?

When using a microcomputer system, programs should be designed around a complete data base. Files should contain, in so far as possible, only one type of information and have a format general enough for a variety of purposes (e.g., nodal point information for use by both stress calculation and graphics mesh plotting subroutines). Single workspaces with dynamic dimensioning (i.e., pointers that change with the problem) are recommended for the ease with which available memory may be managed. Overlay schemes for subroutines should be used to ensure that a minimum amount of memory is taken up by program instructions. In choosing the extent of overlaying to be used, study is required to determine the optimum balance between I/O and computation and the amount of core needed for variable storage.

Never select microcomputer operating system software (regardless of the language in use) which has low limits on the number of user subroutines accessible to main programs. Subroutines or functions should be able to be individually

overlayed even though they may reside in large libraries of many such routines. Beware of eight bit microcomputers whose small word size leads to accuracy problems (when double precision software is not available) and limits mass storage file size (i.e., where the operating system chooses to keep track of total bytes per file).

The Hewlett-Packard System 45 Desktop Computer exhibited only three negative attributes during this study. The first of these was its slow speed (a problem which has no practical solution). The second was the lack of a FORTRAN compiler. The third was the relatively high cost of the HP-9845A compared to other micro- and even mini- computers. Although the System 45 is extremely accurate and capable, it is much in demand for graphics related work (for which it is perhaps best suited) and it is probably not cost effective to devote so expensive a machine to hours of slow and deliberate number crunching. Organizations who wished to use the HP-9845A for finite element calculations would, however, experience few difficulties.

The Apple-II Plus Personal Computer, with the operating system and hardware configuration previously discussed, is not a suitable tool for serious finite element work. The reader will recall that:

(1) The system is too slow, taking over two hours to solve 160 equations having a half-bandwidth of 64.

(2) There is a limit of seven compiler calls ($USES statements) to external units for other subroutines or functions.

(3) Random access files are limited to 32,768 bytes in total length.

(4) The machine word is too small and double precision is not available. This causes a loss of accuracy in practical calculations.

(5) Using more than one implicit function definition per program causes multiple subroutines with the name DUMMY to be generated and results in compile time errors.

(6) Many machine features cannot be used with the FORTRAN language alone.

If the Apple-II must be employed it is recommended that another operating system and central processor chip be used.

The author looks forward in anticipation to the wedding of the finite element method and the microcomputers of the future and to the powerful vehicle of discovery which will then be widely available.

APPENDIX A

SEPARATE COMPILATION OF APPLE FORTRAN PROGRAM SEGMENTS

Due to the small size of the Apple-II Plus memory, it is often necessary to compile larger programs in separate pieces and/or to overlay certain subroutines during execution of a large main program. This is made possible in Apple FORTRAN through the use of compiler directives [9].

Compiler directives are used to communicate information to the compiler via the FORTRAN TEXT file. These statements all begin with a dollar sign in column one and some have special requirements as to their location within a file (e.g. $USES statements must be placed at the beginning of a TEXT file and may only be preceeded by Comment lines). For details about the majority of compiler directives statements see Reference [9]. A brief commentary on some of the statements appears in the table below:

Table 9: Compiler Directives in Apple FORTRAN

$INCLUDE filename
: Tells the compiler when this statement is encountered to immediately compile the contents of the FORTRAN TEXT file filename.

Table 9 (contd)

$XREF
This statement produces a cross-reference listing of the compilation. Note that the extra information generated by this statement reduces the size of programs which may be compiled. If not absolutely required this directive should be avoided.

$EXT [type] FUNCTION name #params
$EXT SUBROUTINE name #params
Used to identify assembly language program. calls and not used in this thesis.

The format of the $USES control statement is:

$USES unitname [IN filename] [OVERLAY]

where brackets enclose optional items and capitalized items, when appearing, must be exactly as shown.

The character string "unitname" is a Pascal term which actually represents a collection of P-code "procedures" or "functions" (FORTRAN subroutines or FORTRAN function subprograms previously compiled on the Apple) with their associated compiler directives. This collection is compiled as a single entity and remains together during any OVERLAY. The great disadvantage in this technique is that more code will be called into core than is actually needed whenever unused subroutines reside in the same unit as one which is desired. If one decides to simply put a single subroutine in each unit to circumvent this problem, there remains a limit on the maximum number of $USES statements which are allowed by the compiler in any one program.

The character string "filename" includes all of the information necessary for the compiler and linker to find the file and usually includes the diskette name or volume number followed by a colon, the file name followed by a per iod and the extension CODE. If the filename is not included the compiler and linker will expect to find the unit in the file #4:SYSTEM.LIBRARY.

The character string OVERLAY tells the compiler and linker that the unit named is to be loaded into core only when called and is to remain in memory only while the procedure of interest is in use.

A maximum of seven $USES statements may appear in any one program or subprogram being compile. Whenever more than this number appear in a file, compile time error number 205 is generated and the system usually "hangs" and must be re-booted. This is a real inconvenience when setting up a program in modular form and assigning all repetitive tasks to subroutine or in setting up a large overlay scheme. For example, of one uses only one level of depth in program development (e.g. the main program calls all subroutines and no subroutine calls any other) then the program may be broken into only seven parts, some of which may not be small enough to allow storage for the desired number of variables to reside in core. If graphics are required in any subroutine, one of the seven $USES statements will probably have been used to link the TURTLEGRAPHICS unit from the

#4:SYSTEM.LIBRARY. For programs where subroutines call or overlay other subroutines the situation is much more complex and is best illustrated by the example in the following section.

Reference [9] discusses the use of the $USES compiler directive when more than one level of depth is used in program development, but gives no actual examples. Since the solution of this problem is so critical to the success of implementing a large program on the Apple-II Plus, it was decided to examine a complex case of subroutine interconnection.

The tables which follow contain actual files which were successfully compiled, linked and tested on the Apple-II. Although the files are real, they are simply presented as a hypothetical case which exercises the compiler and linker to the maximum extent and which illustrates to the reader how to structure a difficult program sequence. Nine subroutines are entered (I, C, III, S1, S2, S3, S4, S5, and IV) from a sequence of seven CODE files (ITOIV, S5, S4, S3, S2, S1, ATOD) on disk THESIS9: using all seven available $USES statements. Note also that as long as no more $USES statements are required, an even greater number of subroutine calls could be present (e.g., S5 could call I, II, and III which are in the same unit, and Pascal "segment," as IV).

Table 10: Contents of Demonstration File THESIS9:COMPILE.TEXT

```
$USES UI IN THESIS9:ITOIV.CODE OVERLAY
$USES US5 IN THESIS9:S5.CODE OVERLAY
$USES US4 IN THESIS9:S4.CODE OVERLAY
$USES US3 IN THESIS9:S3.CODE OVERLAY
$USES US2 IN THESIS9:S2.CODE OVERLAY
$USES US1 IN THESIS9:S1.CODE OVERLAY
$USES UA IN THESIS9:ATOD.CODE OVERLAY
      PROGRAM CMPILE
      CALL I
      CALL C
      CALL S1
      STOP
      END
```

Table 11: Contents of Demonstration File THESIS9:ITOIV.TEXT

```
      SUBROUTINE I
      WRITE(*,'(A)') 'SUBROUTINE I ENTERED'
      RETURN
      END
      SUBROUTINE II
      WRITE(*,'(A)') 'SUBROUTINE II ENTERED'
      RETURN
      END
      SUBROUTINE III
      WRITE(*,'(A)') 'SUBROUTINE III ENTERED'
      RETURN
      END
      SUBROUTINE IV
      WRITE(*,'(A)') 'SUBROUTINE IV ENTERED'
      RETURN
      END
```

Table 12: Contents of Demonstration File THESIS9:ATOD.TEXT

```
$USES UI IN THESIS9:ITOIV.CODE OVERLAY
      SUBROUTINE A
      WRITE(*,'(A)') 'SUBROUTINE A ENTERED'
      WRITE(*,'(A)') 'CALLING SUBROUTINE I'
      CALL I
      RETURN
      END
      SUBROUTINE B
      WRITE(*,'(A)') 'SUBROUTINE B ENTERED'
      WRITE(*,'(A)') 'CALLING SUBROUTINE II'
      CALL II
      RETURN
      END
      SUBROUTINE C
      WRITE(*,'(A)') 'SUBROUTINE C ENTERED'
      WRITE(*,'(A)') 'CALLING SUBROUTINE III'
      CALL III
      RETURN
      END
      SUBROUTINE D
      WRITE(*,'(A)') 'SUBROUTINE D ENTERED'
      WRITE(*,'(A)') 'CALLING SUBROUTINE IV'
      CALL IV
      RETURN
      END
```

Table 13: Contents of Demonstration File THESIS9:S1.TEXT

```
$USES UI IN THESIS9:ITOIV.CODE OVERLAY
$USES US5 IN THESIS9:S5.CODE OVERLAY
$USES US4 IN THESIS9:S4.CODE OVERLAY
$USES US3 IN THESIS9:S3.CODE OVERLAY
$USES US2 IN THESIS9:S2.CODE OVERLAY
      SUBROUTINE S1
      WRITE(*,'(A)') 'SUBROUTINE S1 ENTERED'
      WRITE(*,'(A)') 'CALLING SUBROUTINE S2'
      CALL S2
      RETURN
      END
```

Table 14: Contents of Demonstration File THESIS9:S2.TEXT

```
$USES UI IN THESIS9:ITOIV.CODE OVERLAY
$USES US5 IN THESIS9:S5.CODE OVERLAY
$USES US4 IN THESIS9:S4.CODE OVERLAY
$USES US3 IN THESIS9:S3.CODE OVERLAY
      SUBROUTINE S2
      WRITE(*,'(A)') 'SUBROUTINE S2 ENTERED'
      WRITE(*,'(A)') 'CALLING SUBROUTINE S3'
      CALL S3
      RETURN
      END
```

Table 15: Contents of Demonstration File THESIS9:S3.TEXT

```
$USES UI IN THESIS9:ITOIV.CODE OVERLAY
$USES US5 IN THESIS9:S5.CODE OVERLAY
$USES US4 IN THESIS9:S4.CODE OVERLAY
      SUBROUTINE S3
      WRITE(*,'(A)') 'SUBROUTINE S3 ENTERED'
      WRITE(*,'(A)') 'CALLING SUBROUTINE S4'
      CALL S4
      RETURN
      END
```

Table 16: Contents of Demonstration File THESIS9:S4.TEXT

```
$USES UI IN THESIS9:ITOIV.CODE OVERLAY
$USES US5 IN THESIS9:S5.CODE OVERLAY
      SUBROUTINE S4
      WRITE(*,'(A)') 'SUBROUTINE S4 ENTERED'
      WRITE(*,'(A)') 'CALLING SUBROUTINE S5'
      CALL S5
      RETURN
      END
```

The reader who is developing a large program for the Apple-II Plus is encouraged to try placing the "skeleton" of the anticipated program on the machine before proceeding with final coding. This will confirm that the control flow and overlay structure are possible, but will not prove that the final program will fin into core and leave room for the desired number of variables during execution.

Table 17: Contents of Demonstration File THESIS9:S5.TEXT

```
$USES UI IN THESIS9:ITOIV.CODE OVERLAY
      SUBROUTINE S5
      WRITE(*,'(A)') 'SUBROUTINE S5 ENTERED'
      WRITE(*,'(A)') 'CALLING SUBROUTINE IV'
      CALL IV
      RETURN
      END
```

If it becomes necessary, during program development or as a feature of the final code, to know the amount of free memory available for more instructions or variables, a Pascal program must be used. The reason for this is that the Apple-II Plus is really just a host for the "P-mcahine" or pseudo machine described in Reference [10] and this virtual machine uses Pascal accessible registers to keep track of memory allocation. Appendix B of the above reference contains a memory map which shows that the amount of free memory is defined to be the difference between "KP", which is the top of the program stack (this translates to the highest address

Chapter 15 of Reference [9] attempts to describe use of a Pascal FUNCTION unit by an Apple FORTRAN main program and dives an example Pascal program (complete with two minor, unintentional syntax errors).

In an attempt to make this requirement as painless as possible, the author has written and tested a Pascal program that will provide a FORTRAN function called MEMREM() which the reader may use to find out the amount of free memory remaining. The two tables below provide listings of the Pascal unit and a FORTRAN main program which will illustrate the use of the MEMREM() function. In order to use the function and test program the reader must:

1. Boot the system with the APPLE0: and APPLE1: diskettes in volumes #4: and #5.
2. Enter the E(ditor and I(nsert the Pascal unit as shown in the table below. Q(uit and U(Pdate as usual.
3. C(ompile the resulting TEXT file using the Pascal compiler. The resulting CODE file need not be L(inked.
4. Enter the F(iler and S(ave the TEXT and CODE files to the desired FORTRAN working diskette.
5. Create the FORTRAN program in the usual manner and C(ompile using the FORTRAN compiler after re-booting with FORT1: and FORT2: in volumes #4: and #5.
6. Enter the L(inker and link in the usual manner with the FORTRAN CODE file as host and SYSTEM.LIBRARY and the Pascal CODE files as library files.

Table 18: Contents of Pascal File #11:MEMREM.TEXT

```
(*$P*)
(*$S+,L CONSOLE: *)
UNIT MEMORY;
 INTERFACE
  FUNCTION MEMREM: INTEGER;
   IMPLEMENTATION
    FUNCTION MEMREM;
     BEGIN
      MEMREM:=MEMAVAIL;
     END;
    BEGIN
   (* THIS FUNCTION RETURNS THE AMOUNT OF FREE MEMORY *)
  END.
```

Table 19: Apple FORTRAN Program Example Using the MEMREM() Function

```
      $USES MEMORY IN #11:MEMREM.CODE
      C
            PROGRAM MEMTST
      C
            IMPLICIT REAL(A-H,O-Z, INTEGER(I-N)
            INTEGER WORDS,REALS,BYTES
      C
            WORDS = MEMREM ()
            BYTES = 2 * WORDS
            REALS = WORDS / 2
      C
            WRITE(*,10) BYTES,WORDS,REALS
      C
            STOP
      C
         10 FORMAT(//,' SPACE REMAINING FOR:',/,
           1 4X,1I8,' BYTES OF A PROGRAM OR',/,
           1 4X,1I8,' INTEGER VARIABLES OR',/,
           1 4X,1I8,' REAL VARIABLES !!!')
      C
            END
```

For comparison purposes, the test program given above resulted in a linked FORTRAN CODE file of 27 (512 byte) blocks of diskette storage (the bulk of which was consumed by the Run Time Unit code). When executed the program returned a value of 24512 bytes of program available out of an approximate theoretical maximum, prior to program loading, of 41,000 bytes. This is enough room for 12256 integer variables or 6128 floating point variables.

The reader should also note on the memory map in Reference [10] that both the program and the variables in the data heap appear to be able to write over the two pages of High-Resolution Graphics memory located between 8K and 24K in the Apple-II Plus RAM. The author has not investigated the possible ramifications of this storage scheme, but doubts the validity of the MEMREM() function if high resolution graphics are used in the same program with this function.

SUBROUTINE SOLVE (HP9845A VERSION)

```
10   SUB Solve(Nn,Mm,Ns,File$)
20   ! ************************************************************************
30   !
40   !   ORIGINAL CODE (IBM360/67 FORTRAN): GILLES CANTIN, DECEMBER 1969
50   !   REFERENCE: INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN
60   !              ENGINEERING,VOL. 3, 379-388(1971)
70   !
80   !   RE-CODED FOR HEWLETT-PACKARD SYSTEM 45 (9845A): APRIL 1981
90   !
100  !   PASSING PARAMETERS:
110  !            Mm   IS THE NUMBER OF COLUMNS OF BLOCKS
120  !            Nn   IS THE NUMBER OF ROWS OF BLOCKS
130  !            Ns   IS THE SIZE OF EACH BLOCK (Ns BY Ns ELEMENTS)
140  !         File$   IS THE FILE CONTAINING THE MATRIX TO BE SOLVED
150  !
160  !   IN PRESENT FORM THE SUBROUTINE IS INDEPENDENT OF THE TYPE OF MASS
170  !   STORAGE UTILIZED.  DISK MASS STORAGE (HARD OR FLEXIBLE) IS RE-
180  !   COMMENDED.  OVERLAP I/O MODE IS DECLARED UPON ENTRY TO MINIMIZE
181  !   WAITING TIME FOR READ/WRITE MASS STORAGE OPERATIONS.  SERIAL MODE
190  !   IS DECLARED UPON EXIT TO AVOID DISABLING ERROR TRAPS IN THE CALL-
200  !   ING PROGRAM.
210  !
```

```
220  !   ALL TRANSFERS WITHIN THE SUBROUTINE ARE MADE TO LABELS TO ALLOW
230  !   RE-NUMBERING FOR THE CONVENIENCE OF THE USER.
240  !
250  !   OTHER SUBROUTINES REQUIRED:
260  !            Rddisk, Wrdisk, Mult, Syminv
270  !
280  !   OTHER FUNCTION SUBROUTINES REQUIRED:
290  !            FNTrk
300  !
310  ! ************************************************************************
320  ASSIGN #1 TO File$
330  BUFFER #1
340  OVERLAP
350  OPTION BASE 1
360  DIM Ak1(Ns^2),Ak2(Ns^2),Ak3(Ns^2),Rb1(Ns),Rb2(Ns),Rb3(Ns)
370  N=0
380  Kmm=Mm
390 Sol100:  N=N+1  ! THIS VARIABLE KEEPS TRACK OF THE CURRENT BLOCK ROW
400  Ntk=FNTrk(N,1,Mm)
410  Ntr=N*(Mm+1)
420  CALL Rddisk(#1,Ntk,Ak2(*))
430  CALL Syminv(Ak2(*),Ns,Rb1(*),Rb2(*),Iflg)
```

```
440  IF Iflg=1 THEN Sol600
450  IF Iflg<>2 THEN Sol110
460  PRINT "BLOCK ";N;" IS NEARLY SINGULAR"
470  BEEP
480 Sol110:  CALL Rddisk(#1,Ntr,Rb1(*))
490  CALL Mult(Ak2(*),Rb1(*),Rb2(*),Ns,Ns,1)
500  CALL Wrdisk(#1,Ntr,Rb2(*))
510  IF N=Nm THEN Sol300  ! ARE WE DONE WITH THE LAST ROW OF BLOCKS ?
520  IF Ns<=Nn-Mm+1 THEN Sol120
530  Kmm=Kmm-1  ! FOR THIS ROW, THE FIRST COLUMN THAT WILL HAVE A ZERO BLOCK
540 Sol120:  FOR K=2 TO Kmm  ! THIS LOOP REDUCES THE OTHER BLOCKS IN THIS ROW
550  Ntk=FNTrk(N,K,Mm)
560  CALL Rddisk(#1,Ntk,Ak1(*))
570  CALL Mult(Ak2(*),Ak1(*),Ak3(*),Ns,Ns,Ns)
580  CALL Wrdisk(#1,Ntk,Ak3(*))
590  FOR I=1 TO Ns
600  Ii=(I-1)*Ns
610  FOR J=1 TO Ns
620  Ii=Ii+J
630  Ir=I+(J-1)*Ns
640  Ak3(Ii)=Ak1(Ir)
650  NEXT J
```

```
660  NEXT I
670 Sol150:  Ntk=FNTrk(Nn,K,Mm)
680  CALL Wrdisk(#1,Ntk,Ak3(*))
690 Sol200:  NEXT K
700  FOR L=2 TO Lmm  ! THIS LOOP REDUCES AFFECTED BLOCKS BELOW THE CURRENT ROW
710  I=N+L-1
720  IF I>Nn THEN Sol260
730  J=0
740  Ntk=FNTrk(Nn,L,Mm)
750  CALL Rddisk(#1,Ntk,Ak2(*))
760  FOR K=L TO Kmm
770  J=J+1
780  Ntk=FNTrk(I,K,Mm)
790  CALL Rddisk(#1,Ntk,Ak1(*))
800  CALL Mult(Ak2(*),Ak1(*),Ak3(*),Ns,Ns,Ns)
810  Ntk=FNTrk(I,J,Mm)
820  CALL Rddisk(#1,Ntk,Ak1(*))
830  FOR I1=1 TO Ns^2
840  Ak1(I1)=Ak1(I1)-Ak3(I1)
850 Sol210:  NEXT I1
860  CALL Wrdisk(#1,Ntk,Ak1(*))
870 Sol250:  NEXT K
```

```
860  CALL Mult(Rb2(*),Rb3(*),Rb3(*),Ns,Ns,1)
890  Ntr=I*(Mm+1)
900  CALL Rddisk(#1,Ntr,Rb1(*))
910  FOR I1=1 TO Ns
920  Rb1(I1)=Rb1(I1)-Rb3(I1)
930 Sol255:  NEXT I1
940  CALL Wrdisk(#1,Ntr,Rb1(*))
950 Sol260:  NEXT I
960  GOTO Sol100  ! RETURN TO GET THE NEXT ROW OF BLOCKS
970 Sol300:  N=N-1  ! THIS IS THE BEGINNING OF THE BACK SUBSTITUTION PHASE
980  IF N=0 THEN Sol500  ! ARE WE ALL FINISHED WITH THE SOLUTION ?
990  Nt1=N*(Mm+1)
1000 CALL Rddisk(#1,Nt1,Rb3(*))
1010 FOR K=2 TO Kmm
1020 L=N+K-1
1030 IF L>Nr THEN Sol400
1040 Nt1=FNTr1(N,K,Mm)
1050 CALL Rddisk(#1,Nt1,Ab1(*))
1060 Ntr=L*(Mm+1)
1070 CALL Rddisk(#1,Ntr,Rb1(*))
1080 CALL Mult(Ab1(*),Rb1(*),Rb2(*),Ns,Ns,1)
1090 FOR I1=1 TO Ns
```

```
1100  Rb3(I1)=Rb3(K1)-Rb2(I1)
1110 Sol310:  NEXT I1
1120 Sol400:  NEXT K
1130  CALL Wrdisk(#1,Nr1,Rb3(*))
1140  Kmm=Kmm+1
1150  IF Kmm<=Mm THEN Sol300
1160  Kmm=Mm
1170  GOTO Sol300
1180 Sol500: SERIAL
1190 SUBEXIT
1200 Sol600:  PRINT "BLOCK ";N;" IS SINGULAR"
1210 Sol000:  BEEP
1220  DISP "ABNORMAL TERMINATION"
1230  STOP
1240  SUBEND
```

SUBROUTINE SYMINV (HP9845A VERSION)

```
6000  SUB Syminv(A(*),N,B(*),C(*),Iflg)
6010  OPTION BASE 1
6020  Iflg=0
6030  Mtrace=0
6040  FOR I=1 TO N
6050  Iad=(I-1)*N+I
6060  Mtrace=Mtrace+ABS(A(Iad))
6070 Sym005:  NEXT I
6080  Mpzro=Mtrace*1.0E-9
6090  Lp=N
6100  FOR I=2 TO Lp
6110  FOR J=I TO Lp
6120  Icol=J+Lp*(I-2)
6130  Irow=I+(J-1)*Lp-1
6140  !! A(Icol)=A(Irow) THEN Sym010
6150  A(Icol)=.5*(A(Icol)+A(Irow))
6160  A Irow)-A(Icol)
6170 Sym010:  NEXT J
6180 NEXT I
6190 FOR I=1 TO N
6200 B(I)=0
6210 I1=(I-1)*N
```

```
6220 FOR J=1 TO N
6230 Jj=Ii+J
6240 B(I)=B(I)+ABS(A(Ij))
6250 Sym020: NEXT J
6260 NEXT I
6270 Anr=0
6280 FOR I=1 TO N
6290 Anr=MAX(Anr,B(I))
6300 Sym030: NEXT I
6310 FOR I=1 TO N
6320 Nr=(I-1)*N
6330 FOR J=1 TO N
6340 Ij=Nr+J
6350 B(J)=A(Ij)
6360 Sym100: NEXT J
6370 D=B(I)
6380 IF D=0 THEN Sym180
6390 IF ABS(D)<Apzro THEN Irig=2
6400 FOR J=1 TO N
6410 C(J)=-1*B(J)/D
6420 Sym110: NEXT J
6430 L=1
```

```
6440 FOR J=1 TO N
6450 M=L
6460 FOR I=J TO N
6470 A(L)=A(L)+B(J)*C(I)
6480 Sym115: A(M)=A(L)
6490 M=M+N
6500 L=L+1
6510 Sym120: NEXT K
6520 L=L+J
6530 Sym130: NEXT J
6540 C(I)=-1 N
6550 M 1
 5600 FOR I I TO N
6570 C IN(J
6580 A(L)=C(J)
6590 A(M)=C(J)
6591 M=M+N
6600 NEXT J
6610 Sym140: NEXT I
6620 N=-N*N
6630 FOR J=1 TO N
6640 A(J)=-1*A(J)
```

```
6650 Sym150: NEXT J
6660 FOR I=1 TO N
6670 B(I)=0
6680 Ii=(I-1)*N
6690 FOR J=1 TO N
6700 Jj=Ii+J
6710 B(I)=B(I)+ABS(A(Jj))
6720 NEXT J
6730 Sym160: NEXT I
6740 Anr=0
6750 FOR I=1 TO N
6760 Anr=MAX(Anr,B(I))
6770 Sym170: NEXT I
6780 Cnbr=Anr*Ainr
6781 PRINT ""
6790 PRINT "CONDITION NUMBER ";Cnbr
6800 SUBEXIT
6810 Itlg=1
6820 SUBEND
```

SUBROUTINE MULT (HP9845A VERSION)

```
5000 SUB Mult(A(*),B(*),C(*),Nra,Nca,Ncb)
5010 FOR I=1 TO Nra
5020 FOR J=1 TO Ncb
5030 Ic=I+(J-1)*Nra
5040 Temp=0
5050 FOR K=1 TO Nca
5060 Ia=I+(K-1)*Nra
5070 Ib=K+(J-1)*Nca
5080 Temp=Temp+A(Ia)*B(Ib)
5090 Sum100: NEXT K
5100 C(Ic)=Temp
5110 NEXT J
5120 Sum110: NEXT I
5130 END
```

SUBROUTINE RDDISK (HP9845A VERSION)

```
3000 SUB Rddisk(#1,Recno,A1(*))
3001 ! THIS SUBROUTINE READS RECORD NUMBER Recno FROM FILE #1 INTO MATRIX
3002 ! A1(*).
3010 OPTION BASE 1
3020 READ #1,Recno;A1(*)
3030 SUBEND
```

SUBROUTINE WRDISK (HP-9845A VERSION)

```
4000 SUB Wrdisk(#1,Recno,A1(*))
4001 ! THIS SUBROUTINE WRITES MATRIX A1(*) INTO RECORD NUMBER Recno IN FILE #1.
4010 OPTION BASE 1
4020 PRINT #1,Recno;A1(*)
4030 SUBEND
```

SUBROUTINE ENTRY (HP9845A VERSION)

```
2000  DEF FNTrk(I,J,Mm)
2001  ! THIS FUNCTION RETURNS THE RECORD NUMBER WHERE BLOCK(I,J) IS STORED.
2002  ! Mm IS THE SAME PARAMETER AS IN THE SOLVE ROUTINE.
2010  Trk=(I-1)*(Mm+2)+J-I+1
2020  RETURN Trk
2030  FNEND
```

PROGRAM TEST (HP9845A VERSION)

```
10    INPUT "ENTER THE NUMBER OF ROWS OF BLOCKS:",Nn
20    INPUT "ENTER THE BLOCK HALF-BANDWIDTH:",Mm
30    INPUT "ENTER THE SUBMATRIX BLOCKSIZE:",Ns
40    File$="STIFF:F8,1"
50    CALL Test(Nn,Mm,Ns)
60    CALL Solve(Nn,Mm,Ns,File$)
70    CALL Answer(Nn,Mm,Ns,File$)
80    END
90    SUB Test(Nn,Mm,Ns)
100   OPTION BASE 1
110   DIM A(Ns,Ns),A1(Ns^2),B(Ns)
120   OVERLAP
130   Numblks=Nn*(Mm+1)
140   IF (Numblks<1) OR (Numblks>32767) THEN A
150   Blksize=Ns^2
160   IF Blksize>32767 THEN A
170   Bytrec=Blksize*8
180   IF Bytrec>32767 THEN A
190   Phyrec=Bytrec/256*Numblks
200   IF Phyrec<=1901 THEN A
210   GOTO Tes001
220 A: DISP "IMPROPER PARAMETERS PASSED TO TEST PROGRAM"
```

```
230  GOTO Abort
240 Tes001:  PRINT LIN(1)
250  PRINT "BLOCK SOLVER TEST RUN DATA:"
260  PRINT LIN(1)
270  PRINT "   SUBMATRIX BLOCK SIZE (Ns):      ";Ns
280  PRINT "   NUMBER OF ROWS OF BLOCKS (Nn): ";Nn
290  PRINT "   BLOCK HALF BANDWIDTH (Mm):      ";Mm
291  Numeqn=Nn*Ns
300  PRINT "   NUMBER OF EQUATIONS:            ";Numeqn
301  Truehb=Mm*Ns
310  PRINT "   TRUE HALF-BANDWIDTH:            ";Truehb
320  PRINT "   TOTAL NUMBER OF BLOCKS:         ";Numblks
330  PRINT "   BLOCKSIZE/RECORD LENGTH(Bytes):";Bytrec
350  PRINT LIN(1)
360  CREATE "STIFF:F8,1",Numblks,Bytrec
370  ASSIGN #1 TO "STIFF:F8,1"
380  MAT A=ZER
390  FOR I=1 TO Ns
400  Fv=Ns+1
410  FOR J=I TO Ns
420  IF J>Ns THEN Tes200
430  Fv=Fv-1
```

```
440  A(I,J)=Fv
450  NEXT J
460 Tes200:  NEXT I
470  FOR I=1 TO Ns
480  FOR J=1 TO Ns
490 Tes250:  A(J,I)=A(I,J)
500  NEXT J
510  NEXT I
520  CALL Equiv(A1(*),A(*),Ns)
530  PRINT "THIS IS THE MATRIX THAT WAS PUT IN THE DIAGONAL BLOCKS:"
540  MAT PRINT A1
550  PRINT "DIAGONAL BLOCK/RECORD/TRACK NUMBERS GENERATED AND WRITTEN:"
560  FOR I=1 TO Nn
570  Recno=FNTrk(I,1,Mm)
580  PRINT Recno;
590  PRINT #1,Recno;A1(*)
600  NEXT I
610  PRINT LIN(1)
620  MAT A=IDN
630  MAT A=A*(2)
640  CALL Equiv(A1(*),A(*),Ns)
650  PRINT "THIS IS THE MATRIX THAT WAS PUT IN THE OFF-DIAGONAL BLOCKS:"
```

```
660  MAT PRINT A1
670  PRINT "OFF-DIAGONAL BLOCK/RECORD/TRACK NUMBERS GENERATED AND WRITTEN:"
680  FOR I=1 TO Nn
690  FOR J=2 TO Mm
700  Recno=FNTrk(I,J,Mm)
710  PRINT Recno;
720  PRINT #1,Recno;A1(*)
730  NEXT J
740  NEXT I
750  PRINT LIN(1)
760  FOR I=1 TO Ns
770  B(I)=10.0
780  NEXT I
790  PRINT LIN(2)
800  PRINT "THIS IS THE MATRIX THAT WAS PUT IN THE R.H.S. BLOCKS:"
810  MAT PRINT B
820  PRINT "R. H. S. BLOCK/RECORD/TRACK NUMBERS GENERATED AND WRITTEN:"
830  FOR I=1 TO Nn
840  Recno=I*(Mm+1)
850  PRINT Recno;
860  PRINT #1,Recno;B(*)
870  NEXT I
```

```
880  PRINT LIN(1)
890  SERIAL
900  SUBEXIT
910  SUBEND
920  SUB Equiv(A1(*),A(*),Ns)
930  OPTION BASE 1
940  FOR I=0 TO Ns-1
950  FOR J=1 TO Ns
960  A1(I*Ns+J)=A(I+1,J)
970  NEXT J
980  NEXT I
990  PRINT LIN(2)
1000 SUBEND
```

SUBROUTINE ANSWER (HP9845A VERSION)

```
3420 SUB Answer(Nn,Mm,Ns,File#)
3430 OPTION BASE 1
3440 DIM B(Ns)
3450 ASSIGN #1 TO File#
3460 PRINT LIN(2),"ANSWERS:",LIN(2)
3470 FOR I=1 TO Nn
3480 Recno=I*(Mm+1)
3490 PRINT "FROM BLOCK --->";Recno
3500 READ #1,Recno;B(*)
3510 MAT PRINT B
3520 NEXT I
3530 PRINT LIN(2)
3540 SUBEND
```

APPENDIX C

SAMPLE EQUATION SOLUTION = HEWLETT PACKARD SYSTEM 45

```
BLOCK SOLVER TEST RUN DATA:

   SUBMATRIX BLOCK SIZE (Ns):      8
   NUMBER OF ROWS OF BLOCKS (Nn):  20
   BLOCK HALF BANDWIDTH (Nw):      8
   NUMBER OF EQUATIONS:            160
   TRUE HALF-BANDWIDTH:            64
   TOTAL NUMBER OF BLOCKS:         180
   BLOCKSIZE/RECORD LENGTH(Bytes): 512

THIS IS THE MATRIX THAT WAS PUT IN THE DIAGONAL BLOCKS:
 8 4 7 5 6 6 5 7 4 8 3 2 6 1 5
 7 3 8 4 7 5 6 6 5 7 4 8 3 2 6
 6 2 7 3 8 4 7 5 6 6 5 7 4 8 3 7
 5 1 6 2 7 3 8 4 7 5 6 6 5 7 4 8
```

DIAGONAL BLOCK/RECORD/TRACK NUMBERS GENERATED AND WRITTEN:
1 10 19 28 37 46 55 64 73 82 91 100 109 118 127 136 145 154
163 172

THIS IS THE MATRIX THAT WAS PUT IN THE OFF-DIAGONAL BLOCKS:

2	0	0	0
0	0	0	0
0	2	0	0
0	0	0	0
0	0	2	0
0	0	0	0
0	0	0	2
0	0	0	0
0	0	0	0
2	0	0	0
0	0	0	0
0	2	0	0
0	0	0	0
0	0	2	0
0	0	0	0
0	0	0	2

OFF-DIAGONAL BLOCK/RECORD/TRACK NUMBERS GENERATED AND WRITTEN:
2 3 4 5 6 7 8 11 12 13 14 15 16 17 20 21 22 23 24 25 26 29

```
 30   31   32   33   34   35   38   39   40   41   42   43   44   47   48   49   50   51   52   53
 56   57   58   59   60   61   62   65   66   67   68   69   70   71   74   75   76   77   78   79
 80   83   84   85   86   87   88   89   92   93   94   95   96   97   98  101  102  103  104
 105  106  107  110  111  112  113  114  115  116  119  120  121  122  123  124
 125  128  129  130  131  132  133  134  137  138  139  140  141  142  143  146
 147  148  149  150  151  152  155  156  157  158  159  160  161  164  165  166
 167  168  169  170  173  174  175  176  177  178  179

THIS IS THE MATRIX THAT WAS PUT IN THE R.H.S. BLOCKS:
 10                   10                   10                   10
 10                   10                   10                   10

R. H. S. BLOCK RECORD TRACK NUMBERS GENERATED AND WRITTEN:
  9   18   27   36   45   54   63   72   81   90   99  108  117  126  135  144  153  162
  171  180

CONDITION NUMBER  96.0000000005

CONDITION NUMBER  71.2308192552

CONDITION NUMBER  102.0666643489

CONDITION NUMBER  116.441895794

CONDITION NUMBER  124.736507169

CONDITION NUMBER  130.1092217666
```

CONDITION NUMBER 133.8542622011

CONDITION NUMBER 136.601183783

CONDITION NUMBER 60.0419924713

CONDITION NUMBER 125.85385339

CONDITION NUMBER 2056.02559205

CONDITION NUMBER 235.96754888

CONDITION NUMBER 158.006058614

CONDITION NUMBER 487.7259791

CONDITION NUMBER 342.170017404

CONDITION NUMBER 2913.13227398

CONDITION NUMBER 84.1624662538

CONDITION NUMBER 57.772249913

CONDITION NUMBER 219.581763707

CONDITION NUMBER 68.6303115896

ANSWERS:

FROM [illegible] ---> 9

.28002218935 .14650694380 .15344529569 .11665130695
.11665130727 .15344529569 .19850 [illegible] .28002218935

FROM BLOCK ---> 18
.34587181691 .16644629455 .11097369845 .11338412158
.11338412164 .11097370178 .16644632540 .34587181913

FROM BLOCK ---> 27
.32985603517 .16631338711 .10753889384 .10532248729
.10532249176 .10753888521 .16631338807 .32985603635

FROM BLOCK ---> 36
.31[illegible]254031 .16618044747 .10410408414 .09726085884
.09726086537 .10410408755 .16618044443 .31384025511

FROM BLOCK ---> 45
.37968967205 .13411982455 .06163248861 .09399368214
.09399368730 .16324829418 02 .13411982803 .37968987301

FROM BLOCK ---> 54
-.04[illegible]51038435 .[illegible]49577666 .29[illegible]01[illegible] .10024329371
.10[illegible]4[illegible]96754 .[illegible]3032[illegible] .[illegible]45 -.04[illegible]51038471

FROM BLOCK ---> 63

.21442933269	.04540350611	.00788875165	.29635032165
.29635033132	.00788875434	.04540350200	.21442933339

FROM BLOCK ---> 72

.09444789129	.28012134243	.20399577092	-.03072206401
-.03072206795	.20399576744	.28012135255	.09444788640

FROM BLOCK ---> 81

.11962362100	.09951278322	.13484058429	.17731218651
.17731218266	.13484058990	.09951277740	.11962362390

FROM BLOCK ---> 90

.10838744082	.13589965901	.14409423337	.14752903576
.14752904279	.14409422734	.13589966285	.10838743973

FROM BLOCK ---> 99

.10838743993	.13589966066	.14409423010	.14752904132
.14752903515	.14409423568	.13589965751	.10838744073

FROM BLOCK ---> 108

.11962[illegible]	.0[illegible]	.1[illegible]	.177312182088
.17731216294[illegible]	.1[illegible]40580[illegible]	.09[illegible]	.11[illegible]24581

FROM BLOCK 117

.09[illegible]	.26012134421	.203995774968	-.030722061945
.030[illegible]	.203995778527	.280121336624	.094447882805

FROM BLOCK ---> 126

.21442934243	.045403509806	.007888750438	.296350316886
.296350321684	.007888746932	.045403509663	.21442934212

FROM BLOCK ---> 135

.04695104[illegible]	.203995767839	.296350322988	.100243309535
.100243301441	.296350320658	.20399577742	-.04695105046[illegible]

FROM BLOCK ---> 144

.2[illegible]	.1[illegible]	.06163248798	.09399368144
.0[illegible]	.0[illegible]	.1341198189	.37968986963

FROM BLOCK ---> 153

.313840252857	.16618044495	.104104086[illegible]	.097260858124
.097260861434	.104104081402	.16618044807[illegible]	.313840252944

FROM BLOCK ---> 162

.329856034772	.166313388561	.107538894454	.105322491352
.105322485872	.107538894454	.166313384383	.329856037705

FROM BLOCK ---> 171

.345871817107	.166446327557	.110973699935	.113384118974
.113384121689	.110973696971	.166446330451	.345871816124

FROM BLOCK ---> 180

.280022201234	.198506951025	.15344529444	.116651297849
.116651297849	.153445299452	.198506948662	.280022200[illegible]

```
SUBROUTINE SOLVE (APPLE-II VERSION)                                    PAGE:    1

      SUBROUTINE SOLVE(NN,MM,NS,AK1,AK2,AK3,RB1,RB2,RB3,NRECFY)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
C*****************************************************************************
C
C                  CODE BY: GILLES CANTIN, DECEMBER 1969
C                      FOR: THE IBM 360/67
C
C              MODIFIED BY: D. J. MULHOLLAND, LT, USN, MARCH 1982
C                      FOR: APPLE II+ STANDARD FORTRAN CONFIGURATION
C
C         NN    IS THE NUMBER OF BLOCKS PER COLUMN
C         MM    IS THE NUMBER OF BLOCKS PER ROW
C         NS    IS THE SIZE OF ONE BLOCK
C
C     NRECFY    IS THE NUMBER OF BLOCKS OF THE CHOSEN RECORD SIZE
C               THAT ARE CONTAINED IN ANY SINGLE FILE.
C               THIS IS CALCULATED FROM AN APPLE LIMIT THAT NO
C               FILE MAY CONTAIN MORE THAN 32768 BYTES.
C
C    OTHER SUBROUTINE REQUIRED:
C
C               MULT, SYMINV, RDDISK, WRDISK
C
C     NOTES:    SUBROUTINES WRDISK AND WRDISK ARE MACHINE DEPENDENT
C               AS ARE LINES 2555 AND 2565 OF SOLVE.
C
```

```
C*************************************************************************
C
      DIMENSION AK1(1),AK2(1),AK3(1),RB1(1),RB2(1),
     1RB3(1)
C
      NTRK(I,J)=(I-1)*(MM+2)+J-I+1
C
      LRECS=(NS**2)*4
      NCOUNT=NS*NS
      N=0
      KMM=MM
  100 N=N+1
C
C       REDUCE BLOCK ROW "N"
C
C       1.-REDUCE R.H.S.
C
      NTK=NTRK(N,1)
      NTR=N*(MM+1)
C
      CALL RDDISK(NTK,AK2,NCOUNT,NRECFY)
      CALL SYMINV(AK2,NS,RB1,RB2,IFLG)
C
      IF(IFLG.EQ.1) GO TO 600
      IF(IFLG.EQ.2) WRITE(6,2000) N
C
      CALL RDDISK(NTR,RB1,NS,NRECFY)
      CALL MULT(AK2,RB1,RB2,NS,NS,1)
```

```
      CALL WRDISK(NTR,RB2,NS,NRECFY)
C
C        2.-  CHECK FOR LAST ROW OF BLOCKS
C
      IF(N.EQ.NN) GO TO 300
C
C        3.-  REDUCE BLOCKS IN ROW "N"
C
      IF(N.GT.(NN-MM+1)) KMM=KMM-1

      DO 200 K=2,KMM
C
      NTK=NTRK(N,K)
C
      CALL RDDISK(NTK,AK1,NCOUNT,NRECFY)
      CALL MULT(AK2,AK1,AK3,NS,NS,NS)
      CALL WRDISK(NTK,AK3,NCOUNT,NRECFY)
C
      DO 150 I=1,NS
C
      II=(I-1)*NS
C
      DO 150 J=1,NS
C
      IL=II+J
      IR=I+(J-1)*NS
C
  150 AK3(IL)=AK1(IR)
```

```
C
      NTK=NTRK(NN,K)
C
      CALL WRDISK(NTK,AK3,NCOUNT,NRECFY)
C
  200 CONTINUE
C
C        4.-  REDUCE REMAINING ROWS OF BLOCKS
C
      DO 260 L=2,KMM
C
      I=N+L-1
      IF(I.GT.NN) GO TO 260
      J=0
C
      NTK=NTRK(NN,L)
C
      CALL RDDISK (NTK,AK2,NCOUNT,NRECFY)
C
      DO 250 K=L,KMM
C
      J=J+1
C
      NTK=NTRK(N,K)
C
      CALL RDDISK(NTK,AK1,NCOUNT,NRECFY)
      CALL MULT(AK2,AK1,AK3,NS,NS,NS)
C
```

```
      NTK=NTRK(I,J)
C
      CALL RDDISK(NTK,AK1,NCOUNT,NRECFY)
C
      DO 210 I1=1,NCOUNT
  210 AK1(I1)=AK1(I1)-AK3(I1)
C
      CALL WRDISK(NTK,AK1,NCOUNT,NRECFY)
C
  250 CONTINUE
C
      CALL MULT(AK2,RB2,RB3,NS,NS,1)
C
      NTR=I*(MM+1)
C
      CALL RDDISK(NTR,RB1,NS,NRECFY)
C
      DO 255 I1=1,NS
  255 RB1(I1)=RB1(I1)-RB3(I1)
C
      CALL WRDISK(NTR,RB1,NS,NRECFY)

  260 CONTINUE
C
      GO TO 100
C
C        BACK SUBSTITUTION
C
```

```
  300 N=N-1
C
C         1.-  CHECK FOR FIRST ROW OF BLOCKS
C
      IF(N.EQ.0) GO TO 500
C
C        2.-  CALCULATE BLOCKS OF UNKNOWNS
C
      NT1=N*(MM+1)
C
      CALL RDDISK(NT1,RB3,NS,NRECFY)
C
      DO 400 K=2,KMM
C
      L=N+K-1
      IF(L.GT.NN) GO TO 400
C
      NTK=NTRK(N,K)
C
      CALL RDDISK(NTK,AK1,NCOUNT,NRECFY)
C
      NTR=L*(MM+1)
C
      CALL RDDISK(NTR,RB1,NS,NRECFY)
      CALL MULT(AK1,RB1,RB2,NS,NS,1)
C
      DO 310 K1=1,NS
  310 RB3(K1)=RB3(K1)-RB2(K1)
```

```
C
  400 CONTINUE
C
      CALL WRDISK(NT1,RB3,NS,NRECFY)
C
      KMM=KMM+1
      IF(KMM.GT.MM) KMM=MM
      GO TO 300
C
  500 CONTINUE
C
      RETURN
C
  600 WRITE(6,1000) N
C
C  C L O S E   A L L   F I L E S
C
 2555 DO 2565 I=7,10,1
 2565 CLOSE(I,STATUS='KEEP')
C
      STOP
C
 1000 FORMAT(5X,'BLOCK (',I3,',1) IS SINGULAR')
 2000 FORMAT(5X,'BLOCK (',I3,',1) IS NEARLY SINGULAR')
C
      END
$INCLUDE SCRATCH:MULT.TEXT
$INCLUDE SCRATCH:SYMINV.TEXT
```

```
SUBROUTINE SOLVE (APPLE-II VERSION)

$INCLUDE SCRATCH:WRDISK.TEXT
$INCLUDE SCRATCH:RDDISK.TEXT
```

```
      SUBROUTINE SYMINV(A,N,B,C,IFLG)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      DIMENSION A(1),B(1),C(1)
C
      IFLG=0
      ZRO=0.0E0
      TRACE=ZRO
C
      DO 5 I=1,N
      IAD=(I-1)*N+I
    5 TRACE=TRACE+ABS(A(IAD))
C
      APZRO=TRACE*1.0E-6
      LP=N
C
      DO 10 I=2,LP
      DO 10 J=I,LP
C
      ICOL=J+LP*(I-2)
      IROW=I+(J-1)*LP-1
      IF(A(ICOL).EQ.A(IROW)) GO TO 10
      A(ICOL)=0.5E0*(A(ICOL)+A(IROW))
      A(IROW)=A(ICOL)
C
   10 CONTINUE
C
```

```
      DO 20 I=1,N
C
      B(I)=ZRO
      II=(I-1)*N
C
      DO 20 J=1,N
C
      JJ=II+J
C
   20 B(I)=B(I)+ABS(A(JJ))
C
      ANR=ZRO
C
      DO 30 I=1,N
   30 ANR=AMAX1(ANR,B(I))
C
      DO 140 I=1,N
C
      NR=(I-1)*N
C
      DO 100 J=1,N
      K=NR+J
  100 B(J)=A(K)
C
      D=B(I)
      IF(D.EQ.ZRO) GO TO 180
      IF(ABS(D).LT.APZRO) IFLG=2
C
```

```
      DO 110 J=1,N
  110 C(J)=-B(J)/D
C
      L=1
C
      DO 130 J=1,N
C
      M=L
C
      DO 120 K=J,N
C
      A(L)=A(L)+B(J)*C(K)
C
  115 CONTINUE
C
      A(M)=A(L)
      M=M+N
C
  120 L=L+1
C
  130 L=L+J
C
      C(I)=-1.0E0/D
      M=I
C
      DO 140 J=1,N
C
      K=NR+J
```

```
      A(K)=C(J)
      A(M)=C(J)
C
  140 M=M+N
C
      NS=N*N
C
      DO 150 J=1,NS
  150 A(J)=-A(J)
C
      DO 160 I=1,N
C
      B(I)=ZRO
      II=(I-1)*N
C
      DO 160 J=1,N
      JJ=II+J
  160 B(I)=B(I)+ABS(A(JJ))
C
      AINR=ZRO
C
      DO 170 I=1,N
  170 AINR=AMAX1(AINR,B(I))
C
      CNBR=ANR*AINR
      WRITE(6,2000) CNBR
C
      RETURN
```

```
C
  180 IFLG=1
      RETURN
C
 2000 FORMAT(5X,'CONDITION NUMBER',5X,1E15.5)
C
      END
```

```
      SUBROUTINE MULT(A,B,C,NRA,NCA,NCB)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      DIMENSION A(1),B(1),C(1)
C
      ZRO=0.0E0
C
      DO 110 I=1,NRA
      DO 110 J=1,NCB
C
      IC=I+(J-1)*NRA
      TEMP=ZRO
C
      DO 100 K=1,NCA
C
      IA=I+(K-1)*NRA
      IB=K+(J-1)*NCA
      TEMP=TEMP+A(IA)*B(IB)
C
  100 CONTINUE
C
      C(IC)=TEMP
C
  110 CONTINUE
C
      RETURN
C
      END
```

```
      SUBROUTINE RDDISK(NACTIV,C,NENTRY,NRECFY)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      DIMENSION C(1)
C
      IF(NACTIV.LE.NRECFY)  IUNIT=7
      IF((NACTIV.GT.NRECFY).AND.(NACTIV.LE.(2*NRECFY)))  IUNIT=8
      IF((NACTIV.GT.(2*NRECFY)).AND.(NACTIV.LE.(3*NRECFY)))  IUNIT=9
      IF((NACTIV.GT.(3*NRECFY)).AND.(NACTIV.LE.(4*NRECFY)))  IUNIT=10
C
      NRECRD=NACTIV-(IUNIT-7)*NRECFY
C
      READ(IUNIT,REC=NRECRD,END=1000) (C(K),K=1,NENTRY)
C
      RETURN
C
 1000 CONTINUE
      WRITE(6,1010) NACTIV
C
      DO 1020 IUNIT=7,10,1
 1020 CLOSE(IUNIT,STATUS='KEEP')
C
      STOP
C
 1010 FORMAT(/,5X,' PROGRAM ABORT !!!',/,' END OF FILE ENCOUNTERED DURING
     1 READ OF BLOCK: ',I5,/,' ALL FILES CLOSED ...')
C
      END
```

```
      SUBROUTINE WRDISK(NACTIV,C,NENTRY,NRECFY)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      DIMENSION C(1)
C
      IF(NACTIV.LE.NRECFY) IUNIT = 7
      IF((NACTIV.GT.NRECFY).AND.(NACTIV.LE.(2*NRECFY))) IUNIT = 8
      IF((NACTIV.GT.(2*NRECFY)).AND.(NACTIV.LE.(3*NRECFY))) IUNIT = 9
      IF((NACTIV.GT.(3*NRECFY)).AND.(NACTIV.LE.(4*NRECFY))) IUNIT = 10
C
      NRECRD=NACTIV-(IUNIT-7)*NRECFY
C
      WRITE(IUNIT,REC=NRECRD) (C(K),K=1,NENTRY)
C
      RETURN
C
      END
```

```
      SUBROUTINE TEST(NN,MM,NS,C,NRECFY)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      DIMENSION C(1)
C
      NCOUNT=NS**2
      NEQN = NN * NS
      NTRUHB = MM * NS
      NBLTOT = NN * (MM + 1)
      LRECS = NCOUNT * 4
C
      DO 100 I=1,NCOUNT
  100 C(I)=0.0E0
C
      DO 125 I=1,NS
C
      ENTRY=FLOAT(NS+1)*1.0E0
C
      DO 125 J=I,NS
C
      IJ=(I-1)*NS+J
      JI=(J-1)*NS+I
      ENTRY=ENTRY-1.0E0
      C(IJ)=ENTRY
C
  125 C(JI)=C(IJ)
C
```

```
      WRITE(6,175)
      WRITE(6,176) NS,NN,MM
      WRITE(6,177) NEQN,NTRUHB,NBLTOT,LRECS
      WRITE(6,200)
      WRITE(6,225) (C(I),I=1,NCOUNT)
      WRITE(6,180)
C
      DO 250 I=1,NN
C
      J = 1
C
      NACTIV=(I-1)*(MM+2)+J-I+1
C
      WRITE(6,181) NACTIV

C
      CALL DISKWR(NACTIV,C,NCOUNT,NRECFY)
C
  250 CONTINUE
C
      DO 260 I=1,NS
      DO 260 J=1,NS
C
      IJ=(I-1)*NS+J
      II=(I-1)*NS+I
      C(IJ)=0.0E0
      C(II)=2.0E0
C
```

```
  260 CONTINUE
C
      WRITE(6,270)
      WRITE(6,225) (C(I),I=1,NCOUNT)
      WRITE(6,120)
C
      DO 280 I=1,NN
      DO 280 J=2,MM
C
      NACTIV=(I-1)*(MM+2)+J-I+1
C
      WRITE(6,181) NACTIV
C
      CALL DISKWR(NACTIV,C,NCOUNT,NRECFY)
C
  280 CONTINUE
C
      DO 290 I=1,NS
      C(I)=10.0E0
  290 CONTINUE
C
      WRITE(6,300)
      WRITE(6,225) (C(I),I=1,NS)
      WRITE(6,130)
C
      DO 310 I=1,NN
C
      NACTIV=I*(MM+1)
```

```
C
      WRITE(6,181) NACTIV
C
      CALL DISKWR(NACTIV,C,NS,NRECFY)
C
  310 CONTINUE
C
      RETURN
C
  180 FORMAT(/,3X,'DIAGONAL BLOCK NUMBERS GENERATED AND WRITTEN:',/)
  181 FORMAT(1X,1I5$)
  120 FORMAT(/,3X,'OFF-DIAGONAL BLOCK NUMBERS GENERATED AND WRITTEN:',/)
  130 FORMAT(/,3X,'R. H. S. BLOCK NUMBERS GENERATED AND WRITTEN:',/)
  175 FORMAT(3X,'BLOCK SOLVER TEST RUN DATA:',/,/)
  176 FORMAT(3X,'SUBMATRIX BLOCKSIZE (NS):          ',1I5,/,
     1       3X,'NUMBER OF ROWS OF BLOCKS (NN):    ',1I5,/,
     2       3X,'BLOCK HALF BANDWIDTH (MM):        ',1I5)
  177 FORMAT(3X,'NUMBER OF EQUATIONS:               ',1I5,/,
     1       3X,'TRUE HALF BANDWIDTH:              ',1I5,/,
     2       3X,'TOTAL NUMBER OF BLOCKS:           ',1I5,/,
     3       3X,'BLOCKSIZE/RECORD LENGTH (BYTES): ',1I5,//)
  200 FORMAT(3X,'DIAGONAL BLOCKS:',/)
  225 FORMAT(8(1F9.1,1X))
  270 FORMAT(/,3X,'OFF-DIAGONAL BLOCKS:',/)
  300 FORMAT(/,3X,'R.H.S. MEMBERS:',/)
C
      END
      SUBROUTINE DISKWR(NACTIV,C,NENTRY,NRECFY)
```

```
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      DIMENSION C(1)
C
      IF(NACTIV.LE.NRECFY) IUNIT = 7
      IF((NACTIV.GT.NRECFY).AND.(NACTIV.LE.(2*NRECFY))) IUNIT = 8
      IF((NACTIV.GT.(2*NRECFY)).AND.(NACTIV.LE.(3*NRECFY))) IUNIT = 9
      IF((NACTIV.GT.(3*NRECFY)).AND.(NACTIV.LE.(4*NRECFY))) IUNIT = 10
C
      NRECRD=NACTIV-(IUNIT-7)*NRECFY
C
      WRITE(*,100) NACTIV,IUNIT,NRECRD
C
      WRITE(IUNIT,REC=NRECRD) (C(K),K=1,NENTRY)
C
      RETURN
C
  100 FORMAT(/,' DISKWR ENTERED !!!',/,' BLOCK: ',I5,
     1 ' UNIT: ',I5,' RECORD: ',I5)
C
      END
```

```
      SUBROUTINE ANSWER(NN,MM,NS,C,NRECFY)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      DIMENSION C(1)
C
      WRITE(*,'(A)') ' ANSWER ENTERED !!!'
C
      WRITE(6,25)
C
      DO 50 I=1,NN
C
      NACTIV=I*(MM+1)
C
      CALL DISKRD(NACTIV,C,NS,NRECFY)
C
      WRITE(6,75) NACTIV
      WRITE(6,100) (C(J),J=1,NS)
C
   50 CONTINUE
C
      WRITE(*,'(A)') ' DEPARTING ANSWER !!!'
C
      RETURN
C
   25 FORMAT(/,3X,' ANSWERS:',/)
   75 FORMAT(/,' FROM BLOCK --) ',I5,/)
  100 FORMAT(4(1E12.5,1X))
```

```
C
      END
      SUBROUTINE DISKRD(NACTIV,C,NENTRY,NRECFY)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      DIMENSION C(1)
C
      IF(NACTIV.LE.NRECFY)  IUNIT=7
      IF((NACTIV.GT.NRECFY).AND.(NACTIV.LE.(2*NRECFY)))  IUNIT=8
      IF((NACTIV.GT.(2*NRECFY)).AND.(NACTIV.LE.(3*NRECFY)))  IUNIT=9
      IF((NACTIV.GT.(3*NRECFY)).AND.(NACTIV.LE.(4*NRECFY)))  IUNIT=10
C
      NRECRD=NACTIV-(IUNIT-7)*NRECFY
C
      WRITE(*,900) NACTIV,IUNIT,NRECRD
C
      READ(IUNIT,REC=NRECRD,END=1000) (C(K),K=1,NENTRY)
C
      RETURN
C
 1000 CONTINUE
      WRITE(6,1010) NACTIV
C
      DO 1020 IUNIT=7,10,1
 1020 CLOSE(IUNIT,STATUS='KEEP')
C
      STOP
```

```
C
  900 FORMAT(/,' DISKRD ENTERED !!!',/,' BLOCK: ',1I5,
     1 ' UNIT: ',1I5,' RECORD: ',1I5)
 1010 FORMAT(/,5X,' PROGRAM ABORT !!!',/,' END OF FILE ENCOUNTERED DURING
     1 READ OF BLOCK: ',1I5,/,' ALL FILES CLOSED ...')
C
      END
```

```
$USES UANSWER IN SCRATCH:ANSWER.CODE OVERLAY
$USES USOLVE IN SCRATCH:SOLVE.CODE OVERLAY
$USES UFILES IN SCRATCH:FILES.CODE OVERLAY
$USES UTEST IN SCRATCH:TEST.CODE OVERLAY
C
      PROGRAM THESIS
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*23 FNAME1,FNAME2,FNAME3,FNAME4
C
      COMMON SPACE(3000)
C
      OPEN(*,FILE='CONSOLE:')
C
    5 WRITE(*,'(A)') 'ENTER THE NUMBER OF ROWS OF BLOCKS'
      READ(*,10) NN
      IF(NN.EQ.0) NN = 1
      IF(NN.GE.1) GO TO 15
      GO TO 5
C
   15 WRITE(*,'(A)') 'ENTER THE NUMBER OF COLUMNS OF BLOCKS'
      READ(*,10) MM
      IF(MM.EQ.0) MM = 1
      IF((MM.LE.NN).AND.(MM.GE.1)) GO TO 20
      GO TO 15
C
   20 WRITE(*,'(A)') 'WHAT IS THE SQUARE BLOCKSIZE ?'
```

```
      READ(*,10) NS
      IF(NS.EQ.0) NS = 8
      IF((NS.GE.1).AND.(NS.LE.32)) GO TO 25
      GO TO 20
C
   25 WRITE(*,'(A)') 'WHAT"S THE FIRST FILE ? (DISK):(FILENAME).
     1(FILETYPE)'
      READ(*,'(A)') FNAME1
      IF(FNAME1.EQ.' ') FNAME1 = 'STIFF:1'
   35 WRITE(*,'(A)') 'WHAT"S THE SECOND FILENAME ? (DISK):(FILENAME).
     1(FILETYPE)'
      READ(*,'(A)') FNAME2
      IF(FNAME2.EQ.' ') FNAME2 = 'STIFF:2'
   45 WRITE(*,'(A)') 'WHAT"S THE THIRD FILENAME ? (DISK):(FILENAME).
     1(FILETYPE)'
      READ(*,'(A)') FNAME3
      IF(FNAME3.EQ.' ') FNAME3 = 'STIFF:3'
   55 WRITE(*,'(A)') 'WHAT"S THE FOURTH FILENAME ? (DISK).(FILENAME).
     1(FILETYPE)'
      READ(*,'(A)') FNAME4
      IF(FNAME4.EQ.' ') FNAME4 = 'STIFF:4'
C
      LRECS=(NS**2)*4
C
      OPEN(6,FILE='PRINTER:')
C
      NCOUNT=NS**2
      NAK1=1
```

```
      NAK2=NAK1+NCOUNT
      NAK3=NAK2+NCOUNT
      NRB1=NAK3+NCOUNT
      NRB2=NRB1+NS
      NRB3=NRB2+NS
      NLAST=NRB3+NS
C
      IF(NLAST.GT.3000) GO TO 5000
C
      CALL FILES(NN,MM,LRECS,FNAME1,FNAME2,FNAME3,FNAME4,NRECFY)
      CALL TEST(NN,MM,NS,SPACE(NAK1),NRECFY)
      CALL SOLVE(NN,MM,NS,SPACE(NAK1),SPACE(NAK2),SPACE(NAK3),
     1SPACE(NRB1),SPACE(NRB2),SPACE(NRB3),NRECFY)
      CALL ANSWER(NN,MM,NS,SPACE(NRB3),NRECFY)
C
      STOP
C
 5000 WRITE(*,'(A)') 'STORAGE CAPACITY EXCEEDED !!! PROGRAM ABORT.'
C
      STOP
C
   10 FORMAT(1I2)
C
      END
```

```
      SUBROUTINE FILES(NN,MM,LRECS,FNAME1,FNAME2,FNAME3,FNAME4,NRECFY)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*23 FNAME1,FNAME2,FNAME3,FNAME4
C
      WRITE(*,'(A)') ' FILES ENTERED !!!'
C
      BLTOT=FLOAT(NN*(MM+1))
      BYTOT=BLTOT*FLOAT(LRECS)
      BYMAX=(2.0**15)*4.0
C
      IF(BYTOT.GT.BYMAX) GO TO 1000
C
      NRECFY=INT(BYMAX/(FLOAT(LRECS)*4.0))
C
      OPEN(7,FILE=FNAME1,ACCESS='DIRECT',FORM='UNFORMATTED',
     1STATUS='OLD',RECL=LRECS)
      OPEN(8,FILE=FNAME2,ACCESS='DIRECT',FORM='UNFORMATTED',
     1STATUS='OLD',RECL=LRECS)
      OPEN(9,FILE=FNAME3,ACCESS='DIRECT',FORM='UNFORMATTED',
     1STATUS='OLD',RECL=LRECS)
      OPEN(10,FILE=FNAME4,ACCESS='DIRECT',FORM='UNFORMATTED',
     1STATUS='OLD',RECL=LRECS)
C
      WRITE(*,'(A)') ' DEPARTING FILES !!!'
C
      RETURN
```

```
C
 1000 CONTINUE
C
      WRITE(6,1010) BYMAX,BYTOT
C
      STOP
C
 1010 FORMAT(/,' THERE ARE A MAXIMUM OF',1X,1E12.0,1X,'BYTES',/,
     1' ALLOWED FOR A PROBLEM.  YOU HAVE ATTEMPTED A',/,
     2' PROBLEM REQUIRING',1X,1E12.0,1X,'BYTES.',/,
     3' SOLUTION OF EQUATIONS ABORTED !!!')
C
      END
```

APPENDIX E

SAMPLE EQUATION SOLUTION = APPLE-II PLUS PERSONAL COMPUTER

```
BLOCK SOLVER TEST RUN DATA:

SUBMATRIX BLOCKSIZE (NS):          8
NUMBER OF ROWS OF BLOCKS (NN):    20
BLOCK HALF BANDWIDTH (MM):         8
NUMBER OF EQUATIONS:             160
TRUE HALF BANDWIDTH:              64
TOTAL NUMBER OF BLOCKS:          180
BLOCKSIZE/RECORD LENGTH (BYTES): 256

DIAGONAL BLOCKS:

  8.0    7.0    6.0    5.0    4.0    3.0    2.0    1.0
  7.0    8.0    7.0    6.0    5.0    4.0    3.0    2.0
  6.0    7.0    8.0    7.0    6.0    5.0    4.0    3.0
```

5.0	6.0	7.0	8.0	7.0	6.0	5.0	4.0
4.0	5.0	6.0	7.0	8.0	7.0	6.0	5.0
3.0	4.0	5.0	6.0	7.0	8.0	7.0	6.0
2.0	3.0	4.0	5.0	6.0	7.0	8.0	7.0
1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0

DIAGONAL BLOCK NUMBERS GENERATED AND WRITTEN:

1 10 19 28 37 46 55 64 73 82 91 100 109
118 127 136 145 154 163 172

OFF-DIAGONAL BLOCKS:

2.0	.0	.0	.0	.0	.0	.0	.0
.0	2.0	.0	.0	.0	.0	.0	.0
.0	.0	2.0	.0	.0	.0	.0	.0
.0	.0	.0	2.0	.0	.0	.0	.0
.0	.0	.0	.0	2.0	.0	.0	.0
.0	.0	.0	.0	.0	2.0	.0	.0

.0 .0 .0 .0 .0 .0 2.0 .0

.0 .0 .0 .0 .0 .0 .0 2.0

OFF-DIAGONAL BLOCK NUMBERS GENERATED AND WRITTEN:

2 3 4 5 6 7 8 11 12 13 14 15 16
17 20 21 22 23 24 25 26 29 30 31 32 33
34 35 38 39 40 41 42 43 44 47 48 49 50 51
52 53 56 57 58 59 60 61 62 65 66 67 68
69 70 71 74 75 76 77 78 79 80 83 84 85
86 87 88 89 92 93 94 95 96 97 98 101 102 103
104 105 106 107 110 111 112 113 114 115 116 119 120
121 122 123 124 125 128 129 130 131 132 133 134 137 1
38 139 140 141 142 143 146 147 148 149 150 151 152 155
156 157 158 159 160 161 164 165 166 167 168 169 170
173 174 175 176 177 178 179

R.H.S. MEMBERS:

10.0 10.0 10.0 10.0 10.0 10.0 10.0 10.0

R. H. S. BLOCK NUMBERS GENERATED AND WRITTEN:

9 18 27 36 45 54 63 72 81 90 99 108 117
126 135 144 153 162 171 180

CONDITION NUMBER .96000E+02

CONDITION NUMBER .71231E+02

CONDITION NUMBER .10207E+03

CONDITION NUMBER .11644E+03

CONDITION NUMBER .12474E+03

CONDITION NUMBER .13011E+03

CONDITION NUMBER .13385E+03

CONDITION NUMBER .13660E+03

CONDITION NUMBER .60042E+02

CONDITION NUMBER .12585E+03

CONDITION NUMBER .20560E+04

CONDITION NUMBER .23597E+03

CONDITION NUMBER .15804E+03

CONDITION NUMBER .48754E+03

CONDITION NUMBER .34217E+03

CONDITION NUMBER .31243E+04

CONDITION NUMBER .86782E+02

CONDITION NUMBER .57805E+02

CONDITION NUMBER .21690E+03

CONDITION NUMBER .68632E+02

ANSWERS:

FROM BLOCK --> 9

.27975E+00 .19845E+00 .15354E+00 .11678E+00

.11693E+00 .15348E+00 .19834E+00 .27985E+00

FROM BLOCK --> 18

.34596E+00 .16645E+00 .11091E+00 .11342E+00

.11326E+00 .11100E+00 .16647E+00 .34593E+00

FROM BLOCK --> 27

.32996E+00 .16629E+00 .10752E+00 .10520E+00

.10540E+00 .10752E+00 .16631E+00 .32990E+00

FROM BLOCK --> 36

.31393E+00 .16616E+00 .10404E+00 .97296E-01

.97174E-01 .10410E+00 .16622E+00 .31388E+00

FROM BLOCK --) 45

.37966E+00 .13426E+00 .61589E-01 .93854E-01

.94054E-01 .61673E-01 .13403E+00 .37977E+00

FROM BLOCK --) 54

-.46983E-01 .20397E+00 .29647E+00 .10030E+00

.10002E+00 .29638E+00 .20415E+00 -.47028E-01

FROM BLOCK --) 63

.21453E+00 .45164E-01 .78499E-02 .29660E+00

.29637E+00 .77530E-02 .45467E-01 .21438E+00

FROM BLOCK --) 72

.94469E-01 .28042E+00 .20400E+00 -.31112E-01

-.30759E-01 .20399E+00 .28012E+00 .94653E-01

FROM BLOCK --> 81

.11954E+00 .99401E-01 .13483E+00 .17755E+00

.17732E+00 .13492E+00 .99465E-01 .11948E+00

FROM BLOCK --> 90

.10840E+00 .13588E+00 .14415E+00 .14741E+00

.14765E+00 .14404E+00 .13589E+00 .10840E+00

FROM BLOCK --> 99

.10839E+00 .13591E+00 .14405E+00 .14764E+00

.14742E+00 .14412E+00 .13592E+00 .10837E+00

FROM BLOCK --> 108

.11957E+00 .99620E-01 .13485E+00 .17719E+00

.17740E+00 .13486E+00 .99440E-01 .11966E+00

FROM BLOCK --> 117

.94481E-01 .28000E+00 .20405E+00 -.30626E-01

-.30896E-01 .20400E+00 .28028E+00 .94372E-01

FROM BLOCK --> 126

.21456E+00 .45323E-01 .78041E-02 .29640E+00

.29642E+00 .77691E-02 .45411E-01 .21451E+00

FROM BLOCK --> 135

-.47239E-01 .20419E+00 .29644E+00 .10008E+00

.10045E+00 .29648E+00 .20381E+00 -.47021E-01

FROM BLOCK --> 144

.37967E+00 .13399E+00 .61620E-01 .94208E-01

.93968E-01 .61635E-01 .13409E+00 .37962E+00

FROM BLOCK --> 153

.31385E+00 .16616E+00 .10413E+00 .97191E-01

.97354E-01 .10407E+00 .16616E+00 .31385E+00

FROM BLOCK --> 162

.32994E+00 .16631E+00 .10745E+00 .10536E+00

.10523E+00 .10757E+00 .16632E+00 .32990E+00

FROM BLOCK --> 171

.34594E+00 .16648E+00 .11095E+00 .11325E+00

.11344E+00 .11096E+00 .16643E+00 .34593E+00

FROM BLOCK --> 180

.28005E+00 .19851E+00 .15349E+00 .11670E+00

.11648E+00 .15341E+00 .19862E+00 .28003E+00

```
PROGRAM PROBLM (APPLE-II PLUS)                                    PAGE:    1

C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES UINPUT IN DEVEL:INPUT.CODE OVERLAY
$USES URWCOMN IN DEVEL:RWCOMN.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
      PROGRAM PROBLM
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*23 FNAME1,FNAME2
      CHARACTER*4 SPK(2),TALK,HED(20)
      CHARACTER*1 ANSWER,AFFIRM
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
      COMMON /ADDB/ MBLOCK,NEQ1,NEQL,NEQR,MLA,NBLOCK
      COMMON /FREQIF/ ISTOH,ISTOTE
      COMMON /MDFRDM/ IDOF(6)
      COMMON /VAR/ NG,MODEX
      COMMON /LONGER/ LONG
      COMMON /SPEAK/ SPK,TALK
      COMMON /RWORK/ A(2000)
      COMMON /IWORK/ IA(2000)
C
      DATA AFFIRM/'Y'/
```

```
C
      SPK(1) = 'FRAN'
      SPK(2) = 'ENGL'
C
      ICOM = 7
      IIN = 5
      IOUT = 6
      IRIG = 4
      IDTAP = 3
      ILOAD = 2
      IELMNT = 1
C
      WRITE(*,'(A)') 'PROBLM VERSION 2.0  22 MAR 82'
      WRITE(*,'(A)') 'ENTER (DISK):(FILENAME).TEXT !!!'
      READ(*,'(A)') FNAME1
      WRITE(*,'(A)') '(CONSOLE:$) OR (PRINTER:$) OUTPUT ???'
      READ(*,'(A)') FNAME2

C
      WRITE(*,'(A)') '---) OPENING FILES !!!'
C
       OPEN(3,FILE='#11:IDTAP',ACCESS='SEQUENTIAL',STATUS='NEW',
     1FORM='UNFORMATTED')
       OPEN(5,FILE=FNAME1,ACCESS='SEQUENTIAL',STATUS='OLD',
     1FORM='FORMATTED')
       OPEN(6,FILE=FNAME2,ACCESS='SEQUENTIAL',STATUS='NEW',
     1FORM='FORMATTED')
       OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='NEW',
```

```
     1FORM='UNFORMATTED')
C
      MTOTR = 2000
      MTOTI = 2000
      LONG = 512
      MBLOCK = 64
C
  100 NUMEST = 0
      MAXEST = 0
      NBCEL = 0
      NWK = 0
      NWM = 0
      NWC = 0
      MIDEST = 0
      MA = 0
      IND = 0
      DO 8 I = 1,3
    8 NPAR(I) = 0
      NBRLOD = 0
      KTR = 0
      NEQL = 0
      NEQR = 0
      MLA = 0
      NBLOCK = 0
      ISTOH = 0
      ISTOTE = 0
      NG = 0
      DO 9 I = 1,MTOTI
```

```
    9 IA(I) = 0
      DO 10 I = 1,MTOTR
   10 A(I) = 0
C
      REWIND(IIN)
C
      READ( IIN, 1000, END=800 ) TALK
      IF(TALK.NE.SPK(1)) TALK = SPK(2)
      READ(IIN,1000) (HED(I),I=1,20)
      READ( IIN, 1001 ) NUMNP,(IDOF(I),I=1,6),NUMEG,NLCASE,MODEX,ISTOTE
C
      IF( NUMNP .GT. 0 ) GO TO 750
C
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 3020)
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 3021)

  800 STOP
C
  750 CONTINUE
C
      NDOF = 6
      DO 150 I = 1,6
  150 NDOF = NDOF - IDOF( I )
C
      IF(TALK.EQ.SPK(1)) WRITE( IOUT,2000 ) (HED(I),I=1,20),NUMNP,
     1 NDOF,NUMEG,NLCASE
      IF(TALK.EQ.SPK(2)) WRITE( IOUT,2001 ) (HED(I),I=1,20),NUMNP,
     1 NDOF,NUMEG,NLCASE
```

```
      IF(TALK.EQ.SPK(1)) WRITE( IOUT,2010 ) MODEX, ISTOTE
      IF(TALK.EQ.SPK(2)) WRITE( IOUT,2011 ) MODEX, ISTOTE
C
      NR1 = 1
      NR2 = NR1 + NUMNP
      NR3 = NR2 + NUMNP
      NR4 = NR3 + NUMNP
      NENDR = NR4
C
      NI1 = 1
      NI2 = NI1 + NDOF * NUMNP
      NENDI = NI2
C
      NFIRST = 0
      NLAST = 0
C
      IF( NR4 .GT. MTOTR ) CALL ERROR( N5 - MTOT, 1, 1 )
      IF( NI2 .GT. MTOTI ) CALL ERROR( N5 - MTOT, 1, 2 )
C
      CALL INPUT( IA(NI1), A(NR1), A(NR2), A(NR3), NUMNP, NDOF, NEQ )
C
      NEQ1 = NEQ + 1
C
      REWIND IDTAP
      IF( MODEX .NE. 0 ) WRITE( IDTAP ) ( IA( I ), I = NI1, NI2-1 )
C
      WRITE(*,'(A)') '---> WRITING INTO DATABASE ALL COMMON'
      WRITE(*,'(A)') '     VARIABLES AND ENTIRE WORKSPACE !!!'
```

```
      CALL RWCOMN(2)
C
            CLOSE(3,STATUS='KEEP')
            CLOSE(5,STATUS='KEEP')
            CLOSE(6)
            CLOSE(7,STATUS='KEEP')
            WRITE(*,'(A)') '---> FILES CLOSED, NORMAL TERMINATION.'
C
            WRITE(*,'(A)') 'SKIP DIAGNOSTICS ??? (Y/N)'
            READ(*,'(A)') ANSWER
            IF( ANSWER .EQ. AFFIRM ) GO TO 999
C
            WRITE(*,'(A)') '---> DIAGNOSTICS:'
            WRITE(*,'(A)') '--> ID ARRAY:'
            WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
            WRITE(*,'(A)') '--> X ARRAY:'
            WRITE(*,9898) (I,A(I),I=NR1,NR2-1)
            WRITE(*,'(A)') '--> Y ARRAY:'
            WRITE(*,9898) (I,A(I),I=NR2,NR3-1)
            WRITE(*,'(A)') '--> Z ARRAY:'
            WRITE(*,9898) (I,A(I),I=NR3,NR4-1)
C
        999 STOP
C
       1000 FORMAT( 20A4 )
       1001 FORMAT(1I5,6I1,1I4,3I5)
       2000 FORMAT( '1',/,' ',20A4,/,
           A'   P A R A M E T R E S   D E   C O N T R O L E :          ',/,
```

```
     B' NOMBRE DE POINTS NODAUX . . . . . (NUMNP)  =',I5,/,
     C' NOMBRE DE DEGRES DE LIBERTES PAR NOEUD  . . (NDOF )  =',I5,/,
     D' NOMBRE DE GROUPES D ELEMENT . . . . . (NUMEG)  =',I5,/,
     E' NOMBRE DE CAS DE CHARGEMENT . . . . . (NLCASE) =',I5,/,
     F)
2001 FORMAT( '1',/,' ',20A4,/,
     A'   C O N T R O L   P A R A M E T E R S :                  ',/,
     B' NUMBER OF NODAL POINTS. . . . . (NUMNP)  =',I5,/,
     C' NUMBER OF DEGREES OF FREEDOM PER NODE . . . (NDOF )  =',I5,/,
     D' NUMBER OF ELEMENT GROUPS. . . . . (NUMEG)  =',I5,/,
     E' NUMBER OF LOADING CASES . . . . . (NLCASE) =',I5,/,
     F)
2010 FORMAT(
     G' MODE D EXECUTION DU PROGRAMME . . . . . (MODEX)  =',I5,/,
     H'       .EQ. 0 : VERIFICATION DES DONNEES SEULEMENT          ',/,
     I'       .EQ. 1 : SOLUTION DU PROBLEME                        ',/,
     J' NOMBRE DES TERMES DE LA MATRICE PAR BLOC. . . (ISTOTE) =',I5,/,
     K'       .EQ. 0 : PAR DEFAUT CALCULE PAR LE PROGRAMME BLOCKS  ',/)
2011 FORMAT(
     G' MODE OF PROGRAM EXECUTION . . . . . (MODEX)  =',I5,/,
     H'       .EQ. 0 : VERIFICATION OF INPUT DATA                  ',/,
     I'       .EQ. 1 : SOLUTION OF THE PROBLEM                     ',/,
     J' NUMBER OF COEFFICIENTS PER BLOCK  . . . . . (ISTOTE) =',I5,/,
     K'       .EQ. 0 : CALCULATED BY DEFAULT BY PROGRAM BLOCKS     ',/)
3020 FORMAT(' ERREUR:   NOMBRE DE NOEUDS NUNMP TROUVE NUL !!!      ')
3021 FORMAT(' ERROR:    NUMBER OF NODES EQUAL TO ZERO !!!          ')
8989 FORMAT('IA(',1I5,')= ',1I6)
9898 FORMAT('A(',1I5,')= ',1E13.6)
```

```
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
C
      SUBROUTINE INPUT( ID, X, Y, Z, NUMNP, NDOF, NEQ )
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*4 SPK(2),TALK
C
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /MDFRDM/ IDOF(6)
      COMMON /SPEAK/ SPK,TALK
C
      DIMENSION X(1),Y(1),Z(1),ID(NDOF,1),IDT(6),IDTOLD(6)
C
      CHARACTER*1 IPRC(4), IPR, JPR, IT
      DATA IPRC / ' ', 'A', 'B', 'C' /
      DATA IPR, JPR, IT  / ' ', ' ', ' '  /
C
      DATA ZE, UN, QS / 0.E0, 1.E0, 45.E0 /
C
      IPRINT = 1
      NOLD = 0
      ITOLD = 0
      N = 0
      KN = 0
      KNOLD = 0
C
      DUM = ZE
```

```
      IPR = IPRC( 1 )
      RAD = ATAN( UN ) / QS
C
   10 READ( IIN, 1000 ) IT,N,JPR,(IDT(I),I=1,6),X(N),Y(N),Z(N),KN
C
      IF(IPRINT .EQ. 51) IPRINT = 1
      IF(IPRINT .NE. 1) GO TO 11
C
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2000 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2010 )
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2001 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2011 )
C
   11 WRITE( IOUT, 2002 ) N,(IDT(I),I=1,6),X(N),Y(N),Z(N),KN,IT
C
      IF( N .EQ. 1 ) IPR = JPR
      IF( IT .EQ. IPRC(1) ) GO TO 12
C
      DUM = Z( N ) * RAD
      Z( N ) = Y( N ) * SIN( DUM )
      Y( N ) = Y( N ) * COS( DUM )
C
   12 II = 0
      DO 15 I = 1, NDOF
   13 II = II + 1
      IF( II .LE. 6 ) GO TO 14
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 3000 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 3010 )
```

```
      STOP
C
   14 IF( IDOF(II) .EQ. 1 ) GO TO 13
   15 ID( I, N ) = IDT( II )
C
      IF( NOLD .EQ. 0 ) GO TO 50
C
      II = 0
      DO 20 I = 1, NDOF
   16 II = II + 1
      IF( II .LE. 6 ) GO TO 18
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 3000 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 3010 )
      STOP
C
   18 IF( IDOF(II) .EQ. 1 ) GO TO 16
      IF( (IDT(II).EQ.0) .AND. (IDTOLD(II).LT.0) ) THEN
      ID(I,N) = ID(I,NOLD)
      IDT(II) = IDTOLD(II)
      ENDIF
   20 CONTINUE
C
      IF( KNOLD .EQ. 0 ) GO TO 50
      NUM = ( N - NOLD ) / KNOLD
      NUMN = NUM - 1
      IF( NUMN .LT. 1 ) GO TO 50
C
      XNUM = NUM
```

```
      DX = ( X( N ) - X( NOLD ) ) / XNUM
      IF( IT .EQ. IPRC( 1 ) ) GO TO 21
C
      ROLD = Y( NOLD ) / COS( DUMOLD )
      RNEW = Y( N ) / COS( DUM )
      DR = ( RNEW - ROLD ) / XNUM
      DT = ( DUM - DUMOLD ) / XNUM
C
      GO TO 22
C
   21 DY = ( Y( N ) - Y( NOLD ) ) / XNUM
      DZ = ( Z( N ) - Z( NOLD ) ) / XNUM
C
   22 K = NOLD
      DO 30 J = 1, NUMN
      KK = K
      K = K + KNOLD
      X( K ) = X( KK ) + DX
      IF( IT .EQ. IPRC( 1 ) ) GO TO 26
C
      ROLD = ROLD + DR
      DUMOLD = DUMOLD + DT
      Y( K ) = ROLD * COS( DUMOLD )
      Z( K ) = ROLD * SIN( DUMOLD )
      GO TO 28
C
   26 Y( K ) = Y( KK ) + DY
      Z( K ) = Z( KK ) + DZ
```

```
C
   28 DO 30 I = 1, NDOF
      ID( I, K ) = ID ( I, KK )
   30 CONTINUE
C
   50 NOLD = N
      KNOLD = KN
      DUMOLD = DUM
C
      DO 60 I = 1, 6
   60 IDTOLD( I ) = IDT( I )
C
      IPRINT = IPRINT + 1
C
      IF( N .NE. NUMNP ) GO TO 10
C
      IPRINT = 1
C
      IF( (IPR.EQ.IPRC(2)) .OR. (IPR.EQ.IPRC(4)) ) GO TO 80
C
      DO 75 N = 1, NUMNP
C
      IF(IPRINT .EQ. 51) IPRINT = 1
      IF(IPRINT .NE. 1) GO TO 63
C
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2000 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2010 )
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2003 )
```

```
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2013 )
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2008 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2018 )
C
   63 I = 1
C
      DO 70 II = 1, 6
      IDT( II ) = IDOF( II )
      IF( IDOF( II ) .EQ. 1 ) GO TO 70
      IDT( II ) =  ID( I, N )
      I = I + 1
   70 CONTINUE
C
      WRITE( IOUT, 2005 ) N,(IDT(I),I=1,6),X(N),Y(N),Z(N)
      IPRINT = IPRINT + 1
   75 CONTINUE
C
      IPRINT = 1
C
   80 CONTINUE
      NEQ = 0
      DO 100 N = 1, NUMNP
      DO 100 I = 1, NDOF
      IF( ID(I,N) ) 110, 120, 110
  120 NEQ = NEQ + 1
      ID( I, N ) = NEQ
C
      GO TO 100
```

```
  110 ID( I, N ) = 0
  100 CONTINUE
C
      IF( (IPR.EQ.IPRC(3)) .OR. (IPR.EQ.IPRC(4)) ) RETURN
C
      DO 175 N = 1, NUMNP
C
      IF(IPRINT .EQ. 51) IPRINT = 1
      IF(IPRINT .NE. 1) GO TO 101
C
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2000 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2010 )
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2004 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2014 )
C
  101 I = 1
C
      DO 170 II = 1, 6
      IDT( II ) = 0
C
      IF( IDOF( II ) .EQ. 1 ) GO TO 170
C
      IDT( II ) = ID( I, N )
      I = I + 1
  170 CONTINUE
C
      IPRINT = IPRINT + 1
C
```

```
  175 WRITE( IOUT, 2006 ) N,(IDT(ICK),ICK=1,6)
C
      RETURN
C
 1000 FORMAT( A1, I4, A1, I4, 5I5, 3F10.0, I5 )
 2000 FORMAT('1  D O N N E S   D E S   N O E U D S :',/)
 2010 FORMAT('1  N O D A L   P O I N T   I N F O R M A T I O N :',/)
 2001 FORMAT(' NOEUDS DONNES',/,' NOEUD CONDITIONS AUX LIMITES',12X,
     1'COORDONNEES DES NOEUDS',7X,'PAS COO',/,' NUMERO',
     2'   U    V    W    XX   YY   ZZ',7X,'X',12X,'Y',12X,'Z')
 2011 FORMAT(' GIVEN NODES  ',/,' NODE     BOUNDARY CONDITIONS',12X,
     1' NODAL COORDINATES    ',7X,'PITCH C',/,' NUMBER',
     2'   U    V    W    XX   YY   ZZ',7X,'X',12X,'Y',12X,'Z')
 2002 FORMAT( 1X, I5, 3I4, I5, 2I4, 3E13.6, I4, 3X, A1 )
 2003 FORMAT(' NOEUDS GENERES')
 2013 FORMAT(' GENERATED NODES')
 2004 FORMAT(' NOEUD                NUMEROS D EQUATIONS                ',/,
     1' NUMERO        U      V      W     XX     YY     ZZ ')
 2014 FORMAT(' NODE                 EQUATION NUMBERS                   ',/,
     1' NUMBER        U      V      W     XX     YY     ZZ ')
 2005 FORMAT( 1X, I5, 3I4, I5, 2I4, 3E13.6 )
 2006 FORMAT( 1X, I5, 3X, 5I6,1I6)
 2008 FORMAT(' NOEUD CONDITIONS AUX LIMITES ',12X,'COORDONNEES DES
     1 NOEUDS',/,' NUMERO  U    V    W    XX   YY   ZZ',7X,'X',12X,'Y',12X,
     2 'Z')
 2018 FORMAT(' NODE  BOUNDARY CONDITIONS    ',12X,'NODAL COORDINATE
     1S      ',/,' NUMBER  U    V    W    XX   YY   ZZ',7X,'X',12X,'Y',12X,
     2 'Z')
```

```
 3000 FORMAT(' ERREUR: DEGRE DE LIBERTE PAR NOEUD ETANT MAUVAIS !')
 3010 FORMAT(' ERROR: WRONG NUMBER OF D.O.F. OF FREEDOM PER NODE !')
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
      SUBROUTINE RWCOMN(IRORW)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*4 SPK(2),TALK
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
      COMMON /ADDB/ MBLOCK,NEQ1,NEQL,NEQR,MLA,NBLOCK
      COMMON /FREQIF/ ISTOH,ISTOTE
      COMMON /MDFRDM/ IDOF(6)
      COMMON /VAR/ NG,MODEX
      COMMON /LONGER/ LONG
      COMMON /SPEAK/ SPK,TALK
      COMMON /RWORK/ A(1)
      COMMON /IWORK/ IA(1)
C
      REWIND ICOM
C
      GO TO(1,2) IRORW
C
    1 READ(ICOM) NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NR9,NR10,NR11,NR12,
     ANUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA,IELMNT,ILOAD,IDTAP,
```

```
     BIRIG,IIN,IOUT,IDEC,ICOM,IND,(NPAR(I),I=1,3),NUMEG,MTOTR,MTOTI,
     CNFIRST,NLAST,NBCEL,NEQ1,NEQL,NEQR,MLA,NBLOCK,ISTOH,(SPK(I),I=1,2)
      READ(ICOM) NBRLOD,KTR,ISTOTE,MBLOCK,NENDR,NENDI,
     A NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NI9,NI10,NI11,NI12,
     B(IDOF(I),I=1,6),NG,MODEX,LONG,NLCASE,TALK,NDOF
C
      READ(ICOM) (A(I),I=NR1,NENDR)
      READ(ICOM) (IA(I),I=NI1,NENDI)
C
      RETURN
C
    2 WRITE(ICOM) NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NR9,NR10,NR11,NR12,
     ANUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA,IELMNT,ILOAD,IDTAP,
     BIRIG,IIN,IOUT,IDEC,ICOM,IND,(NPAR(I),I=1,3),NUMEG,MTOTR,MTOTI,
     CNFIRST,NLAST,NBCEL,NEQ1,NEQL,NEQR,MLA,NBLOCK,ISTOH,(SPK(I),I=1,2)
      WRITE(ICOM) NBRLOD,KTR,ISTOTE,MBLOCK,NENDR,NENDI,
     A NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NI9,NI10,NI11,NI12,
     B(IDOF(I),I=1,6),NG,MODEX,LONG,NLCASE,TALK,NDOF
C
      WRITE(ICOM) (A(I),I=NR1,NENDR)
      WRITE(ICOM) (IA(I),I=NI1,NENDI)
C
      RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
      SUBROUTINE ERROR( N, I, ITYPE )
C
      CHARACTER*4 SPK(2),TALK
C
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /SPEAK/ SPK,TALK
C
      GO TO ( 1, 2, 3, 4, 5 ), I
C
    1 IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2000 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2001 )
      GO TO 6
C
    2 IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2010 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2011 )
      GO TO 6
C
    3 IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2020 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2021 )
      GO TO 6
C
    4 IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2030 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2031 )
      GO TO 6
C
    5 IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2040 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2041 )
```

```
    6 IF( (TALK.EQ.SPK(1)) .AND. (ITYPE .EQ. 1) ) THEN
      WRITE( IOUT, 2050 )
      ENDIF
      IF( (TALK.EQ.SPK(1)) .AND. (ITYPE .EQ. 2) ) THEN
      WRITE( IOUT, 2060 )
      ENDIF
      IF( (TALK.EQ.SPK(2)) .AND. (ITYPE .EQ. 1) ) THEN
      WRITE( IOUT, 2051 )
      ENDIF
      IF( (TALK.EQ.SPK(2)) .AND. (ITYPE .EQ. 2) ) THEN
      WRITE( IOUT, 2061 )
      ENDIF
C
      STOP
C
 2000 FORMAT(' PAS ASSEZ DE MEMOIRE POUR LIRE LES NOEUDS')
 2001 FORMAT(' NOT ENOUGH MEMORY TO READ IN THE NODES   ')
 2010 FORMAT(' PAS ASSEZ DE MEMOIRE POUR LIRE LES ELEMENTS')
 2011 FORMAT(' NOT ENOUGH MEMORY TO READ IN THE ELEMENTS  ')
 2020 FORMAT(' PAS ASSEZ DE MEMOIRE POUR DEFINIR LE CHARGEMENT')
 2021 FORMAT(' NOT ENOUGH MEMORY TO READ IN LOADING INFORMATION')
 2030 FORMAT(' PAS ASSEZ DE MEMOIRE POUR FAIRE L ASSEMBLAGE')
 2031 FORMAT(' NOT ENOUGH MEMORY TO ASSEMBLE THE GLOBAL MATRIX')
 2040 FORMAT(' PAS ASSEZ DE MEMOIRE POUR FAIRE LA RESOLUTION')
 2041 FORMAT(' NOT ENOUGH MEMORY TO SOLVE SYSTEM OF EQUATIONS')
 2050 FORMAT(' DEPASSMENT DE MTOTR DE : ',I5)
 2051 FORMAT('    MUST INCREASE MTOTR BY : ',I5)
 2060 FORMAT(' DEPASSMENT DE MTOTI DE : ',I5)
```

```
 2061 FORMAT('     MUST INCREASE MTOTI BY : ',I5)
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES UCOLHT IN DEVEL:COLHT.CODE OVERLAY
$USES URUSS1 IN DEVEL:RUSS1.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
$USES UTRUSS1 IN DEVEL:TRUS1.CODE OVERLAY
$USES UELEMN1 IN DEVEL:ELEMN1.CODE OVERLAY
$USES UELCAL IN DEVEL:ELCAL.CODE OVERLAY
$USES URWCOMN IN DEVEL:RWCOMN.CODE OVERLAY
C
      PROGRAM ELEMS
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*23 FNAME1,FNAME2
      CHARACTER*4 SPK(2),TALK
      CHARACTER*1 ANSWER,AFFIRM
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
      COMMON /ADDB/ MBLOCK,NEQ1,NEQL,NEQR,MLA,NBLOCK
      COMMON /FREQIF/ ISTOH,ISTOTE
      COMMON /MDFRDM/ IDOF(6)
      COMMON /VAR/ NG,MODEX
      COMMON /LONGER/ LONG
```

```
      COMMON /SPEAK/ SPK,TALK
      COMMON /RWORK/ A(2000)
      COMMON /IWORK/ IA(2000)
C
      DATA AFFIRM/'Y'/
C
      WRITE(*,'(A)') 'ELEMENT VERSION 2.0  23 MAR 82'
      WRITE(*,'(A)') 'ENTER (DISK):(FILENAME).TEXT !!!'
      READ(*,'(A)') FNAME1
      WRITE(*,'(A)') '(CONSOLE:$) OR (PRINTER:$) OUTPUT ???'
      READ(*,'(A)') FNAME2
C

      WRITE(*,'(A)') '---) OPENING FILES !!!'
C
      OPEN(1,FILE='#11:IELMNT',ACCESS='SEQUENTIAL',STATUS='NEW',
     1FORM='UNFORMATTED')
      OPEN(5,FILE=FNAME1,ACCESS='SEQUENTIAL',STATUS='OLD',
     1FORM='FORMATTED')
      OPEN(6,FILE=FNAME2,ACCESS='SEQUENTIAL',STATUS='NEW',
     1FORM='FORMATTED')
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='OLD',
     1FORM='UNFORMATTED')
C
      ICOM = 7
C
      WRITE(*,'(A)') '---) READING FROM DATABASE ALL COMMON'
      WRITE(*,'(A)') '      VARIABLES AND ENTIRE WORKSPACE !!!'
```

```
      CALL RWCOMN( 1 )
C
      WRITE(*,'(A)') 'SKIP DIAGNOSTICS ??? (Y/N)'
      READ(*,'(A)') ANSWER
      IF( ANSWER .EQ. AFFIRM ) GO TO 888
C
      WRITE(*,'(A)') '---> DIAGNOSTICS:'
      WRITE(*,'(A)') '--> ID ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
      WRITE(*,'(A)') '--> X ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR1,NR2-1)
      WRITE(*,'(A)') '--> Y ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR2,NR3-1)
      WRITE(*,'(A)') '--> Z ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR3,NR4-1)
C
  888 IND = 1
C
      WRITE(*,'(A)') '---> WRITING ELEMENT INFORMATION'
      WRITE(*,'(A)') '         INTO FILE IELMNT !!!'
      CALL ELCAL
      CLOSE(1,STATUS='KEEP')
C
      CALL ADDRES( IA(NI3), IA(NI2) )
C
      CLOSE(7,STATUS='DELETE')
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='NEW',
     1FORM='UNFORMATTED')
```

```
      WRITE(*,'(A)') '---> WRITING INTO DATABASE ALL COMMON'
      WRITE(*,'(A)') '     VARIABLES AND ENTIRE WORKSPACE !!!'
      CALL RWCOMN( 2 )
C
      CLOSE(5,STATUS='KEEP')
      CLOSE(6)
      CLOSE(7,STATUS='KEEP')
C
      WRITE(*,'(A)') 'SKIP DIAGNOSTICS ??? (Y/N)'
      READ(*,'(A)') ANSWER
      IF( ANSWER .EQ. AFFIRM ) GO TO 999
C
      WRITE(*,'(A)') 'ID ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
C
      WRITE(*,'(A)') 'MHT ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI2,NI3-1)
C
      WRITE(*,'(A)') 'MAXA ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI3,NI4-1)
C
      WRITE(*,'(A)') 'LM ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI4,NI5-1)
C
      WRITE(*,'(A)') 'MATP ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI5,NI6-1)
C
```

```
      WRITE(*,'(A)') 'E ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR4,NR5-1)
C
      WRITE(*,'(A)') 'AREA ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR5,NR6-1)
C
      WRITE(*,'(A)') 'XYZ ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR6,NR7-1)
C
  999 STOP
C
 8989 FORMAT('IA(',I5,')= ',I6)
 9898 FORMAT('A(',I5,')= ',1E15.6)
C
      END
$INCLUDE DEVEL:ADDRES.TEXT
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
      SUBROUTINE ADDRES(MAXA,MHT)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
C
      DIMENSION MAXA( 1 ), MHT( 1 )
C
      NN = NEQ + 1
      DO 20 I = 1,NN
   20 MAXA(I) = 0
C
      MAXA( 1 ) = 1
      MAXA( 2 ) = 2
      MA = 0
      IF(NEQ .EQ. 1) GO TO 100
C
      DO 10 I = 2,NEQ
      IF( MHT( I ) .GT. MA ) MA = MHT( I )
      MAXA( I+1 ) = MAXA( I ) + MHT( I ) + 1
   10 CONTINUE
  100 MA = MA + 1
C
      NWK = MAXA( NEQ+1 ) - MAXA( 1 )
C
      RETURN
C
```

```
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES UCOLHT IN DEVEL:COLHT.CODE OVERLAY
$USES URUSS1 IN DEVEL:RUSS1.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
$USES UTRUSS1 IN DEVEL:TRUS1.CODE OVERLAY
$USES UELEMN1 IN DEVEL:ELEMN1.CODE OVERLAY
C
      SUBROUTINE ELCAL
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*4 SPK(2),TALK,BLOCK,LABEL
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
      COMMON /VAR/ NG,MODEX
      COMMON /SPEAK/ SPK,TALK
      COMMON /RWORK/ A(1)
      COMMON /IWORK/ IA(1)
C
      REWIND IIN
      REWIND IELMNT
C
      BLOCK = 'ELEM'
```

```
C
      WRITE( IOUT, 2010 )
C
  999 READ(IIN,1001,END=888) LABEL
      IF(LABEL .NE. BLOCK) GO TO 999
C
      IF(TALK .EQ. SPK(1)) WRITE( IOUT, 2000 )
      IF(TALK .EQ. SPK(2)) WRITE( IOUT, 2001 )
C
      DO 100 N = 1, NUMEG
C
      NG = N
C
      READ( IIN, 1000 ) (NPAR(I),I=1,3)
C
      CALL ELEMN1
C
      IF(MIDEST.GT.MAXEST) MAXEST = MIDEST
C
      IF(MODEX .NE. 0) WRITE(IELMNT) MIDEST,(NPAR(I),I=1,3),
     1 (A(I),I=NR4,NR7-1),(IA(I),I=NI4,NI6-1)
C
  100 CONTINUE
C
      RETURN
C
  888 IF(TALK .EQ. SPK(1)) WRITE(IOUT,1002) BLOCK
      IF(TALK .EQ. SPK(2)) WRITE(IOUT,1012) BLOCK
```

```
      STOP
C
 1000 FORMAT( 10I5 )
 1001 FORMAT( 1A4 )
 1002 FORMAT(/,' ERROR: ',1A4,' INPUT DATA BLOCK NOT FOUND.',/)
 1012 FORMAT(/,' ERROR: ',1A4,' INPUT DATA BLOCK NOT FOUND.',/)
 2000 FORMAT(' D O N N E E S   D E S   G R O U P E S   D   EL. :',/)
 2001 FORMAT(' I N P U T   F O R   E L E M E N T   G R O U P S :',/)
 2010 FORMAT(1H1)
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES UCOLHT  IN DEVEL:COLHT.CODE OVERLAY
$USES URUSS1  IN DEVEL:RUSS1.CODE OVERLAY
$USES UERROR  IN DEVEL:ERROR.CODE OVERLAY
$USES UTRUSS1  IN DEVEL:TRUS1.CODE OVERLAY
C
      SUBROUTINE ELEMN1
C
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOT,NDOF,NLCASE,NBRLOD,KTR
C
      NPAR1 = NPAR( 1 )
C
      GO TO ( 1, 2, 3 ), NPAR1
C
    1 CALL TRUSS1
      RETURN
C
    2 RETURN
C
    3 RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES UCOLHT IN DEVEL:COLHT.CODE OVERLAY
$USES URUSS1 IN DEVEL:RUSS1.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
C
      SUBROUTINE TRUSS1
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
      COMMON /RWORK/ A(1)
      COMMON /IWORK/ IA(1)
C
      EQUIVALENCE (NPAR(2), NUME), (NPAR(3), NUMMAT)
C
      NI3 = NI2 + NEQ
      NI4 = NI3 + NEQ + 1
      NI5 = NI4 + 6 * NUME
      NI6 = NI5 + NUME
      NEND1 = NI6
C
      NR5 = NR4 + NUMMAT
      NR6 = NR5 + NUMMAT
```

```
      NR7 = NR6 + 6 * NUME
      NENDR = NR7
C
      IF( NR7 .GT. MTOTR ) CALL ERROR( NR7 - MTOTR, 3, 1 )
      IF( NI6 .GT. MTOTI ) CALL ERROR( NI6 - MTOTI, 3, 2 )
C
      MIDEST = 2 * NUMMAT + 2 * ( 6 * NUME ) + NUME + 1
C
      NNNN = NDOF
C
      CALL RUSS1(IA(NI1),A(NR1),A(NR2),A(NR3),A(NR1),IA(NI2),A(NR4),
     1 A(NR5),IA(NI4),A(NR6),IA(NI5),NNNN)
C
      RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES UCOLHT IN DEVEL:COLHT.CODE OVERLAY
C
      SUBROUTINE RUSS1(ID,X,Y,Z,U,MHT,E,AREA,LM,XYZ,MATP,NNNN)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*4 SPK(2),TALK
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /MDFRDM/ IDOF(6)
      COMMON /SPEAK/ SPK,TALK
      COMMON /RWORK/ A(1)
      COMMON /IWORK/ IA(1)
C
      DIMENSION X(1),Y(1),Z(1),ID(NNNN,1),E(1),AREA(1),LM(6,1),
     1 XYZ(6,1),MATP(1),U(1),MHT(1)
C
      EQUIVALENCE (NPAR(1),NPAR1),(NPAR(2),NUME),(NPAR(3),NUMMAT)
C
      ND = 6
C
  300 IPRINT = 5
      IF(TALK.EQ.SPK(1)) WRITE(IOUT,2000) NPAR1,NUME
```

```
      IF(TALK.EQ.SPK(2)) WRITE(IOUT,2001) NPAR1,NUME
      IF(NUMMAT.EQ.0) NUMMAT = 1
      IF(TALK.EQ.SPK(1)) WRITE(IOUT,2010) NUMMAT
      IF(TALK.EQ.SPK(2)) WRITE(IOUT,2011) NUMMAT
      IF(TALK.EQ.SPK(1)) WRITE(IOUT,2020)
      IF(TALK.EQ.SPK(2)) WRITE(IOUT,2021)
C
      DO 10 I = 1,NUMMAT
      READ(IIN,1000) N,E(N),AREA(N)
   10 WRITE(IOUT,2030) N,E(N),AREA(N)
C
      N = 1
  100 READ(IIN,1020) M,II,JJ,MTYP,KG
C
      IF(KG.EQ.0) KG = 1
  120 IF(M.NE.N) GO TO 200
      I = II
      J = JJ
      MTYPE = MTYP
      KKK = KG
C
  200 XYZ(1,N)=X(I)
      XYZ(2,N)=Y(I)
      XYZ(3,N)=Z(I)
      XYZ(4,N)=X(J)
      XYZ(5,N)=Y(J)
      XYZ(6,N)=Z(J)
C
```

```
      MATP(N)=MTYPE
C
      DO 390 L = 1,6
  390 LM(L,N) = 0
C
      DO 400 L = 1,3
      IF(IDOF(L).EQ.1) GO TO 400
      LM(L,N)=ID(L,I)
      LM(L+3,N)=ID(L,J)
  400 CONTINUE
C
      CALL COLHT(MHT,ND,LM(1,N))
C
      IPRINT = IPRINT + 1
C
      IF( (IPRINT .LE. 55) .AND. (IPRINT .NE. 6) ) GO TO 401
C
      IF(TALK.EQ.SPK(1)) WRITE(IOUT,2040)
      IF(TALK.EQ.SPK(2)) WRITE(IOUT,2041)
      IPRINT = 6
C
  401 WRITE(IOUT,2050) N,I,J,MTYPE
C
      IF(N.EQ.NUME) RETURN
      N = N + 1
      I = I + KKK
      J = J + KKK
      IF(N.GT.M) GO TO 100
```

```
      GO TO 120
C
      DO 6 I = N2,N5-1
    6 IA(I) = 0
C
 1000 FORMAT(I5,2F10.0)
 1020 FORMAT(5I5)
 2000 FORMAT(//,48H D E F I N I T I O N   D E S   E L E M E N T S :,//,
     1       16H TYPE D'ELEMENT,13(2H .),17H( NPAR(1) ) . . =,I5/,
     2       34H     EQ.1  ELEMENT DE FERME SIMPLE/,
     3       34H     EQ.2  ELEMENTS NON DISPONIBLE/,
     4       30H     EQ.3  DANS CETTE VERSION /,
     5       20H NOMBRE D'ELEMENTS .,11(2H .),17H( NPAR(2) ) . . =,I5//)
 2001 FORMAT(//36H E L E M E N T   D E F I N I T I O N ///,
     1 14H ELEMENT TYPE ,13(2H .),17H( NPAR(1) ) . . =,I5/,
     2 25H     EQ.1  TRUSS ELEMENTS/,30H     EQ.2  UNAVAILABLE ELEMENT/,
     3 30H     EQ.3  UNAVAILABLE ELEMENT/,
     4 20H NUMBER OF ELEMENTS.,10(2H .),17H( NPAR(2) ) . . =,I5//)
 2010 FORMAT(//50H D E F I N I T I O N   D E S   M A T E R I A U X :,//
     1         33H NOMBRE DE GROUPE DE MATERIAUX ET
     2           /32H DE SECTION DROITE (AIRES)
     3                          5(2H .),17H( NPAR(3) ) . . =,I5//)
 2011 FORMAT(//42H M A T E R I A L   D E F I N I T I O N    ///,
     1 37H NUMBER OF DIFFERENT SETS OF MATERIAL
     2    /32H AND CROSS SECTIONAL CONSTANTS ,
     3    4(2H .),17H( NPAR(3) ) . . =,I5//)
 2020 FORMAT(///2X,6HNUMERO,4X,6HMODULE,6X,14HSECTION DROITE/
     1 1X,9HDE GROUPE,2X,7HD'YOUNG,10X,4HAIRE,/15X,1HE,14X,1HA)
```

```
 2021 FORMAT(///2X,3HSET,7X,6HYOUNGS,6X,15HCROSS-SECTIONAL/
     1  1X,6HNUMBER,5X,7HMODULUS,10X,4HAREA,/15X,1HE,14X,1HA)
 2030 FORMAT(/I5,4X,E12.6,2X,E14.6)
 2040 FORMAT(40H1 I N F O R M A T I O N   S U R   L E S  ,
     1  18H E L E M E N T S :,///
     2  8H ELEMENT ,5X,4HNODE,5X,4HNODE,7X,8HMATERIAL/,
     3  9H NOMBRE-N,6X,1HI,8X,1HJ,4X,16HNUMERO DE GROUPE/)
 2041 FORMAT(1H1,40H E L E M E N T   I N F O R M A T I O N   ///,
     1  8H ELEMENT ,5X,4HNODE,7X,8HMATERIAL/,
     2  9H NUMBER-N,6X,1HI,8X,1HJ,7X,10HSET NUMBER/)
 2050 FORMAT(I5,6X,I5,4X,I5,7X,I5)
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
      SUBROUTINE COLHT (MHT, ND, LM)
C
      IMPLICIT REAL(A-H,O-Z),INTEGER(I-N)
C
      DIMENSION LM( ND ), MHT( 1 )
C
      LS = 32000
      DO 300 I = 1, ND
      IF( LM(I) ) 100, 300, 100
  100 IF( LM(I) - LS ) 200, 300, 300
  200 LS = LM( I )
  300 CONTINUE
C
      DO 400 I = 1, ND
      II = LM(I)
      IF(II.EQ.0) GO TO 400
      ME = II - LS
      IF(ME.GT.MHT(II)) MHT(II)=ME
  400 CONTINUE
C
      RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES URWCOMN IN DEVEL:RWCOMN.CODE OVERLAY
$USES ULOADS IN DEVEL:LOADS.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
C
      PROGRAM LOAD
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*23 FNAME1,FNAME2
      CHARACTER*4 SPK(2),TALK,BLOCK,LABEL
      CHARACTER*1 ANSWER,AFFIRM
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
      COMMON /ADDB/ MBLOCK,NEQ1,NEQL,NEQR,MLA,NBLOCK
      COMMON /FREQIF/ ISTOH,ISTOTE
      COMMON /MDFRDM/ IDOF(6)
      COMMON /VAR/ NG,MODEX
      COMMON /LONGER/ LONG
      COMMON /SPEAK/ SPK,TALK
      COMMON /RWORK/ A(2000)
      COMMON /IWORK/ IA(2000)
C
```

```
      EQUIVALENCE ( NPAR(2), NUME ), ( NPAR(3), NUMMAT )
C
      DATA AFFIRM/'Y'/
C
      WRITE(*,'(A)') 'LOADS VERSION 2.0   23 MAR 82'
      WRITE(*,'(A)') 'ENTER (DISK):(FILENAME).TEXT !!!'
      READ(*,'(A)') FNAME1
      WRITE(*,'(A)') '(CONSOLE:$) OR (PRINTER:$) OUTPUT ???'
      READ(*,'(A)') FNAME2
C
      ICOM = 7
      BLOCK = 'LOAD'
C
      WRITE(*,'(A)') '---> OPENING FILES !!!'
C
      OPEN(2,FILE='#11:ILOAD',ACCESS='SEQUENTIAL',STATUS='NEW',
     1 FORM='UNFORMATTED')
      OPEN(5,FILE=FNAME1,ACCESS='SEQUENTIAL',STATUS='OLD',
     1 FORM='FORMATTED')
      OPEN(6,FILE=FNAME2,ACCESS='SEQUENTIAL',STATUS='NEW',
     1 FORM='FORMATTED')
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='OLD',
     1 FORM='UNFORMATTED')
C
      WRITE(*,'(A)') '---> READING COMMON VARIABLES AND THE'
      WRITE(*,'(A)') '     ENTIRE WORKSPACE FROM DATABASE !!!'
      CALL RWCOMN( 1 )
C
```

```
      WRITE(*,'(A)') 'SKIP DIAGNOSTICS ??? (Y/N)'
      READ(*,'(A)') ANSWER
      IF( ANSWER .EQ. AFFIRM ) GO TO 777
C
      WRITE(*,'(A)') 'DIAGNOSTICS:'
C
      WRITE(*,'(A)') 'ID ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
C
      WRITE(*,'(A)') 'MHT ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI2,NI3-1)
C
      WRITE(*,'(A)') 'MAXA ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI3,NI4-1)
C
      WRITE(*,'(A)') 'LM ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI4,NI5-1)
C
      WRITE(*,'(A)') 'MATP ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI5,NI6-1)
C
      WRITE(*,'(A)') 'E ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR4,NR5-1)
C
      WRITE(*,'(A)') 'AREA ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR5,NR6-1)
C
      WRITE(*,'(A)') 'XYZ ARRAY:'
```

```
      WRITE(*,9898) (I,A(I),I=NR6,NR7-1)
C
  777 NEND = NR1 + 2 * NUMMAT + 6 * NUME
C
      DO 10 I = NR1, NEND
   10 A(I) = A(I+NR4-NR1)
C
      NR2 = NR1 + NUMMAT
      NR3 = NR2 + NUMMAT
      NR4 = NR3 + 6 * NUME
      NR5 = NR4 + NEQ
C
      IF(TALK .EQ. SPK(1)) WRITE( IOUT, 2010 )
      IF(TALK .EQ. SPK(2)) WRITE( IOUT, 2011 )
C
      REWIND IIN
      REWIND ILOAD
C
  999 READ(IIN,1001,END=888) LABEL
      IF(LABEL .NE. BLOCK) GO TO 999
C
      DO 300 L = 1, NLCASE
C
      READ( IIN, 1010 ) LL, NLOAD
C
      IF(TALK .EQ. SPK(1)) WRITE( IOUT, 2020 ) LL, NLOAD
      IF(TALK .EQ. SPK(2)) WRITE( IOUT, 2021 ) LL, NLOAD
      IF( LL .EQ. L ) GO TO 250
```

```
      IF(TALK .EQ. SPK(1)) WRITE( IOUT, 3000 )
      IF(TALK .EQ. SPK(2)) WRITE( IOUT, 3001 )
      STOP
  250 CONTINUE
C
      NI7 = NI6 + NLOAD
      NI8 = NI7 + NLOAD
      NENDI = NI8
C
      NR6 = NR5 + NLOAD
      NENDR = NR6
C
      IF ( NR6 .GT. MTOTR ) CALL ERROR( NR6 - MTOTR, 3, 1 )
      IF ( NI8 .GT. MTOTI ) CALL ERROR( NI8 - MTOTI, 3, 2 )
C
      CALL LOADS(A(NR4),IA(NI6),IA(NI7),A(NR5),IA(NI1),NLOAD,NDOF,NEQ)
C
  300 CONTINUE
C
      CLOSE(2,STATUS='KEEP')
      CLOSE(5,STATUS='KEEP')
      CLOSE(6)
C
      CLOSE(7,STATUS='DELETE')
C
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='NEW',
     1 FORM='UNFORMATTED')
      WRITE(*,'(A)') '---> WRITING COMMON VARIABLES AND THE'
```

```
      WRITE(*,'(A)') '      ENTIRE WORKSPACE INTO DATABASE !!!'
      CALL RWCOMN( 2 )
C
      CLOSE(7,STATUS='KEEP')
C
      WRITE(*,'(A)') 'SKIP DIAGNOSTICS ??? (Y/N)'
      READ(*,'(A)') ANSWER
      IF( ANSWER .EQ. AFFIRM ) GO TO 555
C
      WRITE(*,'(A)') 'DIAGNOSTICS:'
C
      WRITE(*,'(A)') 'ID ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
C
      WRITE(*,'(A)') 'MHT ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI2,NI3-1)
C
      WRITE(*,'(A)') 'MAXA ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI3,NI4-1)
C
      WRITE(*,'(A)') 'LM ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI4,NI5-1)
C
      WRITE(*,'(A)') 'MATP ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI5,NI6-1)
C
      WRITE(*,'(A)') 'NOD ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI6,NI7-1)
```

```
C
      WRITE(*,'(A)') 'IDIRN ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI7,NI8-1)
C
      WRITE(*,'(A)') 'E ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR1,NR2-1)
C
      WRITE(*,'(A)') 'AREA ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR2,NR3-1)
C
      WRITE(*,'(A)') 'XYZ ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR3,NR4-1)
C
      WRITE(*,'(A)') 'R ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR4,NR5-1)
C
      WRITE(*,'(A)') 'FLOAD ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR5,NR6-1)
C
  555 STOP
C
  888 IF(TALK .EQ. SPK(1)) WRITE(IOUT,1002) BLOCK
      IF(TALK .EQ. SPK(2)) WRITE(IOUT,1012) BLOCK
      STOP
C
 1001 FORMAT(1A4)
 1002 FORMAT(/,' ERROR: ',1A4,' INPUT DATA BLOCK NOT FOUND !!!',/)
 1012 FORMAT(/,' ERROR: ',1A4,' INPUT DATA BLOCK NOT FOUND !!!',/)
```

```
 1010 FORMAT( 2I5 )
 2010 FORMAT('1  C H A R G E M E N T S   D O N N E S :')
 2011 FORMAT('1  L O A D I N G   I N F O R M A T I O N :')
 2020 FORMAT(/,/,' CAS DE CHARGEMENT NUMERO. . . . . .=',I5,/,
     A           ' NOMBRE DE CHARGES CONCENTREES . . . . . .=',I5)
 2021 FORMAT(/,/,' LOADING CASE NUMBER . . . . . .=',I5,/,
     A           ' NUMBER OF CONCENTRATED LOADS. . . . . .=',I5)
 3000 FORMAT(' *** ERREUR: CAS DE CHARGEMENT SANS ORDRE !!!')
 3001 FORMAT(' *** ERROR: LOADING CASE OUT OF ORDER !!!')
 9999 FORMAT('IA(',I5,')= ',I6)
 9898 FORMAT('A(',I5,')= ',1E15.6)
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
      SUBROUTINE LOADS( R, NOD, IDIRN, FLOAD, ID, NLOAD, NDOF, NEQ )
C
      IMPLICIT REAL( A-H, O-Z ), INTEGER( I-N )
C
      CHARACTER*4 SPK(2),TALK
C
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /MDFRDM/ IDOF(6)
      COMMON /VAR/ NG,MODEX
      COMMON /SPEAK/ SPK,TALK
C
      DIMENSION R(NEQ), NOD(NLOAD), IDIRN(NLOAD), FLOAD(NLOAD),
     1 ID(NDOF,1)
      DATA ZE / 0.E0 /
C
      IF(TALK .EQ. SPK(1))WRITE( IOUT, 2000 )
      IF(TALK .EQ. SPK(2)) WRITE( IOUT, 2001 )
      READ( IIN, 1000 ) ( NOD(I), IDIRN(I), FLOAD(I), I = 1, NLOAD )
C
      WRITE( IOUT, 2010 ) ( NOD(I), IDIRN(I), FLOAD(I), I = 1, NLOAD )
C
      IF( MODEX .EQ. 0 ) RETURN
C
      DO 210 I = 1, NEQ
  210 R( I ) = ZE
C
      DO 220 L = 1, NLOAD
```

```
      LN = NOD(L)
      LI = IDIRN( L )
      LM = LI
      DO 230 I = 1, LI
  230 LM = LM - IDOF( I )
      II = ID( LM, LN )
C
      IF( II ) 220, 250, 240
C
  240 R( II ) = R( II ) + FLOAD( L )
C
  220 CONTINUE
C
      WRITE( ILOAD ) (R(K),K = 1, NEQ)
C
      RETURN
C
  250 IF(TALK .EQ. SPK(1)) WRITE( IOUT, 3000 ) LI, LN
      IF(TALK .EQ. SPK(2)) WRITE( IOUT, 3001 ) LI, LN
      STOP
C
 1000 FORMAT( 2I5, F10.0 )
 2000 FORMAT( 31H  NOEUD     D. L.          CHARGE ,/,
     1        31H  NUMERO    NUMERO         VALEUR )
 2001 FORMAT( 31H  NODE      D.O.F.          LOAD  ,/,
     1        31H  NUMBER    NUMBER         VALUE  )
 2010 FORMAT( 1X, I5, 6X, I1, 7X, 1E13.6 )
 3000 FORMAT(' D. L. NOMERO ',I2,' DU NOEUD ',I5,' EST IMPOSE !! ')
```

```
 3001 FORMAT(' DOF   NUMBER ',I2,' OF NODE  ',I5,' IS IMPOSED !! ')
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES URWCOMN IN DEVEL:RWCOMN.CODE OVERLAY
$USES USBLOCK IN DEVEL:SBLOCK.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
C
      PROGRAM BLOCKS
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*1 ANSWER,AFFIRM
      CHARACTER*4 SPK(2),TALK
      CHARACTER*23 FNAME1
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
      COMMON /ADDB/ MBLOCK,NEQ1,NEQL,NEQR,MLA,NBLOCK
      COMMON /FREQIF/ ISTOH,ISTOTE
      COMMON /MDFRDM/ IDOF(6)
      COMMON /VAR/ NG,MODEX
      COMMON /LONGER/ LONG
      COMMON /SPEAK/ SPK,TALK
      COMMON /RWORK/ A(2000)
      COMMON /IWORK/ IA(2000)
C
```

```
      EQUIVALENCE ( NPAR(2), NUME )
C
      DATA AFFIRM/'Y'/
C
      WRITE(*,'(A)') 'BLOCKS VERSION 2.0   24 MAR 82'
      WRITE(*,'(A)') '(CONSOLE:$) OR (PRINTER:$) OUTPUT !!!'
      READ(*,'(A)') FNAME1
C
      ICOM = 7
C
      WRITE(*,'(A)') '---) OPENING FILES !!!'
C
      OPEN(6,FILE=FNAME1,ACCESS='SEQUENTIAL',STATUS='NEW',
     1 FORM='FORMATTED')
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='OLD',
     1 FORM='UNFORMATTED')
C
      WRITE(*,'(A)') '---) READING FROM DATABASE ALL COMMON'
      WRITE(*,'(A)') '     VARIABLES AND ENTIRE WORKSPACE !!!'
      CALL RWCOMN(1)
C
      WRITE(*,'(A)') 'SKIP DIAGNOSTICS ??? (Y/N)'
      READ(*,'(A)') ANSWER
      IF(ANSWER .EQ. AFFIRM) GO TO 888
C
      WRITE(*,'(A)') 'DIAGNOSTICS:'
C
      WRITE(*,'(A)') 'ID ARRAY:'
```

```
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
C
      WRITE(*,'(A)') 'MHT ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI2,NI3-1)
C
      WRITE(*,'(A)') 'MAXA ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI3,NI4-1)
C
      WRITE(*,'(A)') 'LM ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI4,NI5-1)
C
      WRITE(*,'(A)') 'MATP ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI5,NI6-1)
C
      WRITE(*,'(A)') 'E ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR1,NR2-1)
C
      WRITE(*,'(A)') 'AREA ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR2,NR3-1)
C
      WRITE(*,'(A)') 'XYZ ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR3,NR4-1)
C
  888 CONTINUE
C
      NEND = NI1 +NEQ + (NEQ+1) + 6 * NUME + NUME
C
      DO 350 I = NI1,NEND
```

```
  350 IA( I ) = IA( I + NI2 - NI1 )
C
      MSTORE = NEQ1 + 3 * NEQ + MAXEST + NBCEL
C
      NI2 = NI1 + NEQ
      NI3 = NI2 + NEQ + 1
      NI4 = NI3 + 6 * NUME
      NI5 = NI4 + NUME
      NI6 = NI5 + NEQ
      NI7 = NI6 + NEQ
C
      IF( NI7 .GT. MTOTI) CALL ERROR( NI7 - MTOTI, 4, 2 )
C
      IBLOCK = 4
      NBLOCK = 1
C
  400 MELST = IBLOCK * 2
      ISTORL = ( MTOT - MSTORE - MELST )
      IF( ISTOTE .GT. 0 ) ISTORL = ISTOTE
C
      CALL SBLOCK(IA(NI2),IA(NI5),IA(NI6),ISTORL,NBLOCK,NEQ,NWK,ISTOH)
C
      IF( ISTOTE .GT. 0 ) GO TO 450
C
      IF( NBLOCK .LE. IBLOCK ) GO TO 450
C
      IBLOCK = IBLOCK * 2
C
```

```
      IF( IBLOCK .LT. MBLOCK ) GO TO 400
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 3010 ) MBLOCK
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 3011 ) MBLOCK
      STOP
C
  450 MAM = NWK / NEQ + 1
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2030 ) NEQ,NWK,MA,MAM,ISTOH,NBLOCK
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2031 ) NEQ,NWK,MA,MAM,ISTOH,NBLOCK
C
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2040 )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2041 )
C
      NN = NI5 + NBLOCK - 1
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2050 ) (I, I = 1, NBLOCK )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2051 ) (I, I = 1, NBLOCK )
C
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2060 ) ( IA( I ), I = NI5, NN )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2061 ) ( IA( I ), I = NI5, NN )
C
      NN = NI6 + NBLOCK - 1
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2070 ) ( IA( I ), I = NI6, NN )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2071 ) ( IA( I ), I = NI6, NN )
C
      NN = NI5 + NBLOCK
C
      DO 500 I = 1, NBLOCK
  500 IA( NN + I - 1 ) = IA ( NI6 + I - 1 )
C
```

```
      NI6 = NN
      NI7 = NI6 + NBLOCK
      NENDI = NI7
C
      CLOSE(7,STATUS='DELETE')
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='NEW',
     1 FORM='UNFORMATTED')
C
      WRITE(*,'(A)') '---> WRITING INTO DATABASE ALL COMMON'
      WRITE(*,'(A)') '      VARIABLES AND ENTIRE WORKSPACE !!!'
      CALL RWCOMN(2)
C
      CLOSE(7,STATUS='KEEP')
      CLOSE(6)
C
      WRITE(*,'(A)') 'SKIP DIAGNOSTICS ??? (Y/N)'
      READ(*,'(A)') ANSWER
      IF(ANSWER .EQ. AFFIRM) GO TO 999
C
      WRITE(*,'(A)') 'DIAGNOSTICS:'
C
      WRITE(*,'(A)') 'MHT ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
C
      WRITE(*,'(A)') 'MAXA ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI2,NI3-1)
C
      WRITE(*,'(A)') 'LM ARRAY:'
```

```
      WRITE(*,8989) (I,IA(I),I=NI3,NI4-1)
C
      WRITE(*,'(A)') 'MATP ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI4,NI5-1)
C
      WRITE(*,'(A)') 'NCOLBV ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI5,NI6-1)
C
      WRITE(*,'(A)') 'ICOPL ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI6,NI7-1)
C
      WRITE(*,'(A)') 'E ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR1,NR2-1)
C
      WRITE(*,'(A)') 'AREA ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR2,NR3-1)
C
      WRITE(*,'(A)') 'XYZ ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR3,NR4-1)
C
C
  999 STOP
C
 2030 FORMAT(47H1 CHARACTERISTIQUES DU SYSTEME D'EQUATIONS:          /,
     A        47H- NOMBRE D'EQUATIONS  . . . . . . . . . (NEQ) =,I7,/,
     B        47H0 NOMBRE D'ELEMENTS DE LA MATRICE . . . (NWK) =,I7,/,
     C        47H0 DEMI LARGEUR MAXIMALE DE LA BANDE . . (MA)  =,I7,/,
     D        47H0 DEMI LARGEUR CONSIDER&E DE LA BANDE . (MAM) =,I7,/,
```

```
     E       47H0 NOMBRE DE TERMES PAR BLOC . . . . . .(ISTOH)=,I7,/,
     F       47H0 NOMBRE DE BLOCS DE SOLUTION . . . . .(NBLOC)=,I7  )
 2031 FORMAT(47H1 CHARACTERISTICS OF THE SYSTEM OF EQUATIONS:      /,
     A       47H- NUMBER OF EQUATIONS . . . . . (NEQ) =,I7,/,
     B       47H0 NUMBER OF COEFFICIENTS IN THE MATRIX  (NWK) =,I7,/,
     C       47H0 MAXIMUM HALF-BANDWIDTH  . . . . . (MA)  =,I7,/,
     D       47H0 AVERAGE HALF-BANDWIDTH  . . . . . (MAM) =,I7,/,
     E       47H0 NUMBER OF COEFFICIENTS PER BLOCK  . .(ISTOH)=,I7,/,
     F       47H0 NUMBER OF BLOCKS FOR THE SOLUTION . .(NBLOC)=,I7  )
 2040 FORMAT(53H- NOMBRE DE COLONNES PAR BLOC ET 1ER BLOC COUPLE      )
 2041 FORMAT(53H- NUMBER OF COLUMNS PER BLOC AND 1ST COUPLED BLOCK    )
 2050 FORMAT(/31H0 NUMEROS DES BLOCS:                  /, ( 10I6, / ) )
 2051 FORMAT(/31H0 BLOCK NUMBERS:                      /, ( 10I6, / ) )
 2060 FORMAT(/32H0 NOMBRES DE COLONNES DES BLOCS:      /, ( 10I6, / ) )
 2061 FORMAT(/32H0 NUMBER OF COLUMNS PER BLOCK:        /, ( 10I6, / ) )
 2070 FORMAT(/31H0 PREMIERS BLOCS COUPLES:             /, ( 10I6, / ) )
 2071 FORMAT(/31H0 FIRST COUPLED BLOCK:                /, ( 10I6, / ) )
 3010 FORMAT('*** ERREUR: NOMBRE DE BLOCS PLUS GRAND QUE MBLOCK =',I5)
 3011 FORMAT('*** ERROR: NUMBER OF BLOCKS GREATER THAN MBLOCK   =',I5)
 8989 FORMAT('IA(',1I5,')= ',1I6)
 9898 FORMAT('A(',1I5,')= ',1E15.6)
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
      SUBROUTINE SBLOCK(MAXA,NCOLBV,ICOPL,ISTORL,NBLOCK,NEQ,NWK,ISTOH)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*4 SPK(2),TALK
C
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /SPEAK/ SPK,TALK
C
      INTEGER MAXA( 1 ), NCOLBV ( 1 ), ICOPL ( 1 )
C
      IF( NBLOCK .GT. 1 ) GO TO 5
      IF( NWK .GT. ISTORL ) GO TO 5
      NBLOCK = 1
      NCOLBV( 1 ) = NEQ
      ICOPL( 1 ) = 1
      ISTOH = NWK
      RETURN
C
    5 ISTOH = ISTORL / 2
      ISTORL = ISTOH * 2
      DO 10 I = 1, NEQ
      ICL = MAXA( I + 1 ) - MAXA( I )
      IF( ISTOH .GE. ICL ) GO TO 10
      IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2000 ) I
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2001 ) I
      STOP
```

```
C
   10 CONTINUE
C
      NBLOCK = 0
      NN = 0
      IB = 0
C
      DO 100 I = 2, NEQ
  140 II = ISTOH - MAXA( I + 1 ) + 1 + NN
      IF( II ) 120, 100, 100
C
  120 NN = MAXA( I ) - 1
      NBLOCK = NBLOCK + 1
      NCOLBV( NBLOCK ) = I - 1 - IB
      IB = I - 1
      GO TO 140
C
  100 CONTINUE
      NBLOCK = NBLOCK + 1
      NCOLBV( NBLOCK ) = NEQ - IB
C
      DO 50 I = 1, NBLOCK
   50 ICOPL( I ) = I
C
      IF( NBLOCK .EQ. 1 ) RETURN
C
      NN = NCOLBV( 1 )
      DO 200 N = 2, NBLOCK
```

```
      ICLM = 0
      NCOLB = NCOLBV( N )
C
      DO 110 I = 1, NCOLB
      ICL = MAXA( NN + I + 1 ) - MAXA( NN + I ) - I - 1
      IF( ICL .GT. ICLM ) ICLM = ICL
C
  110 CONTINUE
C
      J = N - 1
  150 IF( ICLM .LE. 0 ) GO TO 180
      ICOPL( N ) = J
      ICLM = ICLM - NCOLBV( J )
      J = J - 1
      GO TO 150
C
  180 NN = NN + NCOLBV( N )
  200 CONTINUE
C
      RETURN
C
 2000 FORMAT('*** ERREUR: MEMOIRE CENTRALE PETITE POUR LA COLONNE ',I5 )
 2001 FORMAT('*** ERROR:  NOT ENOUGH MEMORY FOR COLUMN ',I5 )
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES UADDBAN IN DEVEL:ADDBAN.CODE OVERLAY
$USES URUSS2 IN DEVEL:RUSS2.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
$USES UTRUSS2 IN DEVEL:TRUS2.CODE OVERLAY
$USES UELEMN2 IN DEVEL:ELEMN2.CODE OVERLAY
$USES UASSEM IN DEVEL:ASSEM.CODE OVERLAY
$USES URWCOMN IN DEVEL:RWCOMN.CODE OVERLAY
C
      PROGRAM ASEMBL
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*1 ANSWER,AFFIRM
      CHARACTER*4 SPK(2),TALK
      CHARACTER*23 FNAME1
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
      COMMON /ADDB/ MBLOCK,NEQ1,NEQL,NEQR,MLA,NBLOCK
      COMMON /FREQIF/ ISTOH,ISTOTE
      COMMON /MDFRDM/ IDOF(6)
      COMMON /VAR/ NG,MODEX
      COMMON /LONGER/ LONG
```

```
      COMMON /SPEAK/ SPK,TALK
      COMMON /RWORK/ A(2000)
      COMMON /IWORK/ IA(2000)
C
      DATA AFFIRM/'Y'/
C
      ICOM = 7
C
      WRITE(*,'(A)') 'ASSEMBLE VERSION 2.0   26 MAR 82'
      WRITE(*,'(A)') '(CONSOLE:$) OR (PRINTER:$) ERROR MSGS. ???'
      READ(*,'(A)') FNAME1
C
      WRITE(*,'(A)') '---> OPENING SEQUENTIAL FILES !!!'
C
      OPEN(1,FILE='#11:IELMNT',ACCESS='SEQUENTIAL',STATUS='OLD',
     1 FORM='UNFORMATTED')
      OPEN(6,FILE=FNAME1,ACCESS='SEQUENTIAL',STATUS='NEW',
     1 FORM='FORMATTED')
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='OLD',
     1 FORM='UNFORMATTED')
C
      WRITE(*,'(A)') '---> READING FROM DATABASE ALL COMMON'
      WRITE(*,'(A)') '     VARIABLES AND ENTIRE WORKSPACE !!!'
C
      CALL RWCOMN( 1 )
C
      WRITE(*,'(A)') 'SKIP DIAGNOSTICS ??? (Y/N)'
      READ(*,'(A)') ANSWER
```

```
      IF(ANSWER .EQ. AFFIRM) GO TO 777
C
      WRITE(*,'(A)') 'DIAGNOSTICS:'
C
      WRITE(*,'(A)') 'MHT ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
C
      WRITE(*,'(A)') 'MAXA ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI2,NI3-1)
C
      WRITE(*,'(A)') 'LM ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI3,NI4-1)
C
      WRITE(*,'(A)') 'MATP ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI4,NI5-1)
C
      WRITE(*,'(A)') 'NCOLBV ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI5,NI6-1)
C
      WRITE(*,'(A)') 'ICOPL ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI6,NI7-1)
C
      WRITE(*,'(A)') 'E ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR1,NR2-1)
C
      WRITE(*,'(A)') 'AREA ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR2,NR3-1)
C
```

```
      WRITE(*,'(A)') 'XYZ ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR3,NR4-1)
C
      WRITE(*,'(A)') 'R ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR4,NR5-1)
C
  777 IF( MODEX .EQ. 0 ) GO TO 444
C
      WRITE(*,'(A)') '---> OPENING DIRECT ACCESS FILE !!!'
C
      OPEN(4,FILE='#12:IRIG',ACCESS='DIRECT',STATUS='NEW',
     1 FORM='UNFORMATTED',RECL=LONG)
C
  444 NR2 = NR1 + ISTOH
C
      NEND = NI3 + 2 * NBLOCK
C
      DO 350 I = NI3, NEND
  350 IA(I) = IA( I + NI5 - NI3 )
C
      NI4 = NI3 + NBLOCK
      NI5 = NI4 + NBLOCK
C
      IF( MODEX .NE. 0 ) GO TO 550
      GO TO 650
C
  550 CONTINUE
      IND = 2
```

```
      NBCEL = 0
      CALL ASSEM( IA(NI2), A(NR1), IA(NI3), ISTOH )
C
      NBRLOD = 0
      KTR = 1
C
  650 CLOSE(7,STATUS='DELETE')
C
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='NEW',
     1 FORM='UNFORMATTED')
C
      WRITE(*,'(A)') '---> WRITING INTO DATABASE ALL COMMON'
      WRITE(*,'(A)') '     VARIABLES AND ENTIRE WORKSPACE !!!'
C
      CALL RWCOMN( 2 )
C
      WRITE(*,'(A)') '---> CLOSING ALL FILES !!!'
C
      CLOSE(1,STATUS='KEEP')
      IF( MODEX .NE. 0 ) CLOSE(4,STATUS='KEEP')
      CLOSE(6)
      CLOSE(7,STATUS='KEEP')
C
      WRITE(*,'(A)') 'SKIP DIAGNOSTICS ??? (Y/N)'
      READ(*,'(A)') ANSWER
      IF(ANSWER .EQ. AFFIRM) GO TO 999
C
      WRITE(*,'(A)') 'DIAGNOSTICS:'
```

```
C
      WRITE(*,'(A)') 'MHT ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
C
      WRITE(*,'(A)') 'MAXA ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI2,NI3-1)
C
      WRITE(*,'(A)') 'NCOLBV ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI3,NI4-1)
C
      WRITE(*,'(A)') 'ICOPL ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI4,NI5-1)
C
      WRITE(*,'(A)') 'LM ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI5,NI6-1)
C
      WRITE(*,'(A)') 'MATP ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI6,NI7-1)
C
      WRITE(*,'(A)') 'AA ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR1,NR2-1)
C
      WRITE(*,'(A)') 'E ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR2,NR3-1)
C
      WRITE(*,'(A)') 'AREA ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR3,NR4-1)
C
```

```
      WRITE(*,'(A)') 'XYZ ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR4,NR5-1)
C
  999 STOP
C
 8989 FORMAT('IA(',I5,')= ',I16)
 9898 FORMAT('A(',I5,')= ',1E15.6)
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES UADDBAN IN DEVEL:ADDBAN.CODE OVERLAY
$USES URUSS2 IN DEVEL:RUSS2.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
$USES UTRUSS2 IN DEVEL:TRUS2.CODE OVERLAY
$USES UELEMN2 IN DEVEL:ELEMN2.CODE OVERLAY
C
      SUBROUTINE ASSEM( MAXA, AA, NCOLBV, ISTOH )
      IMPLICIT REAL(A-H,O-Z),INTEGER(I-N)
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /ADDB/ MBLOCK,NEQ1,NEQL,NEQR,MLA,NBLOCK
      COMMON /LONGER/ LONG
      COMMON /RWORK/ A(1)
      COMMON /IWORK/ IA(1)
C
      INTEGER MAXA( 1 ), NCOLBV( 1 )
C
      DIMENSION AA(ISTOH)
C
      DATA ZE / 0.E0 /
C
      NEQL = 1
      NEQR = 0
      MLA = 0
```

```
      IADR = 1
      INAD = ( ISTOH + LONG - 1 ) / LONG
C
      DO 300 L = 1, NBLOCK
      NEQR = NEQR + NCOLBV( L )
      DO 100 I = 1, ISTOH
  100 AA( I ) = ZE
C
      REWIND IELMNT
C
      DO 200 N = 1, NUMEG
C
      NG = N
C
      WRITE(*,'(A)') '--) READING ELEMNT INFORMATION '
      WRITE(*,'(1A29,1I3)') '    FOR ELEMNT GROUP NUMBER: ',N
C
      READ( IELMNT ) NUMEST,(NPAR(I),I=1,3)
C
      CALL ELEMN2
C
  200 CONTINUE
C
      WRITE(*,8989) L
      WRITE( IRIG,REC=IADR ) ( AA(ICOUNT), ICOUNT = 1, ISTOH )
      IADR = IADR + INAD
      NEQL = NEQL + NCOLBV( L )
      MLA = MAXA( NEQL ) - 1
```

```
  300 CONTINUE
C
      RETURN
C
 8989 FORMAT('WRITING STIFFNESS RECORD --) ',1I3)
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES UADDBAN IN DEVEL:ADDBAN.CODE OVERLAY
$USES URUSS2 IN DEVEL:RUSS2.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
$USES UTRUSS2 IN DEVEL:TRUS2.CODE OVERLAY
C
      SUBROUTINE ELEMN2
C
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
C
      NPAR1 = NPAR( 1 )
C
      GO TO ( 1, 2, 3 ), NPAR1
C
    1 CALL TRUSS2
      RETURN
C
    2 RETURN
C
    3 RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES UADDBAN IN DEVEL:ADDBAN.CODE OVERLAY
$USES URUSS2 IN DEVEL:RUSS2.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
C
      SUBROUTINE TRUSS2
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /RWORK/ A(1)
      COMMON /IWORK/ IA(1)
C
      EQUIVALENCE (NPAR(2),NUME),(NPAR(3),NUMMAT)
C
      NR3 = NR2 + NUMMAT
      NR4 = NR3 + NUMMAT
      NR5 = NR4 + 6 * NUME
      NENDR = NR5
C
      NI6 = NI5 + 6 * NUME
      NI7 = NI6 + NUME
      NENDI = NI7
C
      IF(NR5 .GT. MTOTR) CALL ERROR(NR5-MTOTR,4,1)
```

```
      IF(NI7 .GT. MTOTI) CALL ERROR(NI7-MTOTI,4,2)
C
      READ(IELMNT) (A(I),I=NR2,NR5-1),(IA(I),I=NI5,NI7-1)
C
      CALL RUSS2(IA(NI1),A(NR2),A(NR3),IA(NI5),A(NR4),IA(NI6))
C
      RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES UADDBAN IN DEVEL:ADDBAN.CODE OVERLAY
C
      SUBROUTINE RUSS2(MHT,E,AREA,LM,XYZ,MATP)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /RWORK/ A(1)
      COMMON /IWORK/ IA(1)
C
      DIMENSION E(1),AREA(1),LM(6,1),XYZ(6,1),MATP(1),MHT(1),S(21),
     1 ST(6),D(3)
C
      EQUIVALENCE (NPAR(2),NUME)
C
      ND = 6
C
  610 DO 500 N = 1,NUME
      MTYPE = MATP(N)
      XL2=0.
      DO 505 L = 1,3
      D(L) = XYZ(L,N)-XYZ(L+3,N)
  505 XL2=XL2+D(L)*D(L)
      XL=SQRT(XL2)
```

```
      XX=E(MTYPE)*AREA(MTYPE)*XL
      DO 510 L= 1,3
      ST(L) = D(L)/XL2
  510 ST(L+3) = -ST(L)
C
      KL=0
      DO 600 L = 1,6
      YY = ST(L)*XX
      DO 600 K = L,6
      KL=KL+1
      S(KL)=ST(K)*YY
  600 CONTINUE
      CALL ADDBAN( A(NR1), IA(NI2), S, LM(1,N), ND )
  500 CONTINUE
C
      RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
C
      SUBROUTINE ADDBAN( A, MAXA, S, LM, ND )
C
      IMPLICIT REAL(A-H,O-Z),INTEGER(I-N)
C
      DIMENSION A(1), S(1),MAXA(1),LM(1)
C
      COMMON /ADDB/ MBLOCK,NEQ1,NEQL,NEQR,MLA,NBLOCK
C
      NDI = 0
C
      DO 400 I = 1, ND
      II = LM( I )
      IF( (II .LT. NEQL) .OR. (II .GT. NEQR) ) GO TO 400
C
      MI = MAXA( II ) - MLA
      KS = I
C
      DO 300 J = 1, ND
      JJ = LM( J )
      IF( JJ ) 300, 300, 100
  100 IJ = II - JJ
C
      IF( IJ ) 300, 200, 200
  200 KK = MI + IJ
      KSS = KS
      IF( J .GE. I ) KSS = J + NDI
```

```
      A( KK ) = A( KK ) + S( KSS )
  300 KS = KS + ND - J
C
  400 NDI = NDI + ND - I
C
      RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES URWCOMN IN DEVEL:RWCOMN.CODE OVERLAY
$USES UCOLSOL IN DEVEL:COLSOL.CODE OVERLAY
$USES ULOADV IN DEVEL:LOADV.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
      PROGRAM SOLVE
C
      IMPLICIT REAL(A-H,O-Z),INTEGER(I-N)
C
      CHARACTER*1 ANSWER,AFFIRM
      CHARACTER*4 SPK(2),TALK
      CHARACTER*23 FNAME1
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
      COMMON /ADDB/ MBLOCK,NEQ1,NEQL,NEQR,MLA,NBLOCK
      COMMON /FREQIF/ ISTOH,ISTOTE
      COMMON /MDFRDM/ IDOF(6)
      COMMON /VAR/ NG,MODEX
      COMMON /LONGER/ LONG
      COMMON /SPEAK/ SPK,TALK
      COMMON /RWORK/ A(2000)
      COMMON /IWORK/ IA(2000)
C
```

```
      EQUIVALENCE ( NPAR(2), NUME )
C
      DATA AFFIRM/'Y'/
C
      ICOM = 7
C
      WRITE(*,'(A)') 'SOLVE VERSION 2.0   25 APRIL 82'
      WRITE(*,'(A)') '(CONSOLE:$) OR (PRINTER:$) ERROR MSGS. ?'
      READ(*,'(A)') FNAME1
C
      WRITE(*,'(A)') '---> OPENING SEQUENTIAL FILES !!!'
C
      OPEN(1,FILE='#11:IELMNT',ACCESS='SEQUENTIAL',STATUS='OLD',
     1 FORM='UNFORMATTED')
      OPEN(2,FILE='#11:ILOAD',ACCESS='SEQUENTIAL',STATUS='OLD',
     1 FORM='UNFORMATTED')
      OPEN(3,FILE='#11:IDTAP',ACCESS='SEQUENTIAL',STATUS='OLD',
     1 FORM='UNFORMATTED')
      OPEN(6,FILE=FNAME1,ACCESS='SEQUENTIAL',STATUS='NEW',
     1 FORM='FORMATTED')
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='OLD',
     1 FORM='UNFORMATTED')
C
      CALL RWCOMN( 1 )
C
      IF(MODEX .EQ. 0) THEN
      WRITE(*,'(A)') '---> UNABLE TO RUN SOLVE !!!'
      WRITE(*,'(A)') '    ( MODEX EQUAL TO ZERO)'
```

```
      CLOSE(1,STATUS='KEEP')
      CLOSE(2,STATUS='KEEP')
      CLOSE(3,STATUS='KEEP')
      CLOSE(6)
      CLOSE(7,STATUS='KEEP')
      GO TO 999
      ENDIF
C
      IF( KTR .GT. 1 ) THEN
C
      WRITE(*,'(A)') 'SKIP ENTRY DIAGNOSTICS ??? (Y/N)'
      READ(*,'(A)') ANSWER
      IF(ANSWER .EQ. AFFIRM) GO TO 444
C
      WRITE(*,'(A)') 'MULTIPLE LOADCASE DIAGNOSTICS:'
C
      WRITE(*,'(A)') 'MAXA ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
C
      WRITE(*,'(A)') 'NCOLBV ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI2,NI3-1)
C
      WRITE(*,'(A)') 'ICOPL ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI3,NI4-1)
C
      WRITE(*,'(A)') 'ID ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI4,NI5-1)
C
```

```
      WRITE(*,'(A)') 'LM ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI5,NI6-1)
C
      WRITE(*,'(A)') 'MATP ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI6,NI7-1)
C
      WRITE(*,'(A)') 'A ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR1,NR2-1)
C
      WRITE(*,'(A)') 'B ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR2,NR3-1)
C
      WRITE(*,'(A)') 'D ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR3,NR4-1)
C
      WRITE(*,'(A)') 'V ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR4,NR5-1)
C
      WRITE(*,'(A)') 'E ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR5,NR6-1)
C
      WRITE(*,'(A)') 'AREA ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR6,NR7-1)
C
      WRITE(*,'(A)') 'XYZ ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR7,NR8-1)
C
      ENDIF
```

```
C
  444 IF( KTR .GT. 1 ) GO TO 777
C
      WRITE(*,'(A)') 'SKIP ENTRY DIAGNOSTICS ??? (Y/N)'
      READ(*,'(A)') ANSWER
      IF(ANSWER .EQ. AFFIRM) GO TO 333
C
      WRITE(*,'(A)') 'FIRST LOADCASE DIAGNOSTICS:'
C
      WRITE(*,'(A)') 'MHT ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
C
      WRITE(*,'(A)') 'MAXA ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI2,NI3-1)
C
      WRITE(*,'(A)') 'NCOLBV ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI3,NI4-1)
C
      WRITE(*,'(A)') 'ICOPL ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI4,NI5-1)
C
      WRITE(*,'(A)') 'LM ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI5,NI6-1)
C
      WRITE(*,'(A)') 'MATP ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI6,NI7-1)
C
      WRITE(*,'(A)') 'AA ARRAY:'
```

```
      WRITE(*,9898) (I,A(I),I=NR1,NR2-1)
C
      WRITE(*,'(A)') 'E ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR2,NR3-1)
C
      WRITE(*,'(A)') 'AREA ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR3,NR4-1)
C
      WRITE(*,'(A)') 'XYZ ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR4,NR5-1)
C
  333 NEND = NI1 + (NEQ+1) + 2 * NBLOCK
C
      DO 555 I = NI1,NEND
  555 IA(I) = IA( I + NI2 - NI1 )
C
      NI2 = NI1 + (NEQ+1)
      NI3 = NI2 + NBLOCK
      NI4 = NI3 + NBLOCK
      NI5 = NI4 + NDOF * NUMNP
      NENDI = NI5
C
      NR2 = NR1 + ISTOH
      NR3 = NR2 + ISTOH
      NR4 = NR3 + NEQ
      NR5 = NR4 + NEQ
      NENDR = NR5
C
```

```
      IF( NR5 .GT. MTOTR ) CALL ERROR( NR5 - MTOTR, 5, 1 )
      IF( NI5 .GT. MTOTI ) CALL ERROR( NI5 - MTOTI, 5, 2 )
C
  777 WRITE(*,'(A)') '---> OPENING DIRECT ACCESS FILES !!!'
C
      OPEN(4,FILE='#12:IRIG',ACCESS='DIRECT',STATUS='OLD',
     1 RECL=LONG,FORM='UNFORMATTED')
C
      IF( KTR .GT. 1 ) GO TO 888
C
      CALL COLSOL( IA(NI1), IA(NI2), IA(NI3), A(NR1), A(NR2), A(NR3),
     1  A(NR4), NEQ, NBLOCK, ISTOH, IRIG, IRIG, KTR )
C
      REWIND IDTAP
      NN = NI5 - 1
      READ( IDTAP ) ( IA( I ), I = NI4, NN )
C
      KTR = 2
      IND = 3
C
  888 NBRLOD = NBRLOD + 1
C
      IF( NBRLOD .GT. NLCASE ) THEN
      IF(TALK .EQ. SPK(1)) WRITE (IOUT, 3000)
      IF(TALK .EQ. SPK(2)) WRITE (IOUT, 3001)
      CLOSE(1,STATUS='KEEP')
      CLOSE(2,STATUS='KEEP')
      CLOSE(3,STATUS='KEEP')
```

```
      CLOSE(4,STATUS='KEEP')
      CLOSE(6)
      CLOSE(7,STATUS='KEEP')
      GO TO 999
      ENDIF
C
      REWIND ILOAD
C
      DO 123 I = 1, NBRLOD
  123 CALL LOADV( A(NR4), NEQ )
C
      CALL COLSOL( IA(NI1), IA(NI2), IA(NI3), A(NR1), A(NR2), A(NR3),
     1  A(NR4), NEQ, NBLOCK, ISTOH, IRIG, IRIG, KTR )
C
      CLOSE(1,STATUS='KEEP')
      CLOSE(2,STATUS='KEEP')
      CLOSE(3,STATUS='KEEP')
      CLOSE(4,STATUS='KEEP')
      CLOSE(6)
      CLOSE(7,STATUS='DELETE')
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='NEW',
     1 FORM='UNFORMATTED')
C
      CALL RWCOMN( 2 )
C
      CLOSE(7,STATUS='KEEP')
C
      WRITE(*,'(A)') 'SKIP EXITING DIAGNOSTICS ??? (Y/N)'
```

```
      READ(*,'(A)') ANSWER
      IF(ANSWER .EQ. AFFIRM) GO TO 999
C
      WRITE(*,'(A)') 'EXIT CONDITION DIAGNOSTICS:'
C
      WRITE(*,'(A)') 'MAXA ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
C
      WRITE(*,'(A)') 'NCOLBV ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI2,NI3-1)
C
      WRITE(*,'(A)') 'ICOPL ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI3,NI4-1)
C
      WRITE(*,'(A)') 'ID ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI4,NI5-1)
C
      WRITE(*,'(A)') 'A ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR1,NR2-1)
C
      WRITE(*,'(A)') 'B ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR2,NR3-1)
C
      WRITE(*,'(A)') 'D ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR3,NR4-1)
C
      WRITE(*,'(A)') 'V ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR4,NR5-1)
```

```
      READ(*,'(A)') ANSWER
      IF(ANSWER .EQ. AFFIRM) GO TO 999
C
      WRITE(*,'(A)') 'EXIT CONDITION DIAGNOSTICS:'
C
      WRITE(*,'(A)') 'MAXA ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI1,NI2-1)
C
      WRITE(*,'(A)') 'NCOLBV ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI2,NI3-1)
C
      WRITE(*,'(A)') 'ICOPL ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI3,NI4-1)
C
      WRITE(*,'(A)') 'ID ARRAY:'
      WRITE(*,8989) (I,IA(I),I=NI4,NI5-1)
C
      WRITE(*,'(A)') 'A ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR1,NR2-1)
C
      WRITE(*,'(A)') 'B ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR2,NR3-1)
C
      WRITE(*,'(A)') 'D ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR3,NR4-1)
C
      WRITE(*,'(A)') 'V ARRAY:'
      WRITE(*,9898) (I,A(I),I=NR4,NR5-1)
```

```
C
  999 STOP
C
 8989 FORMAT('IA(',1I5,')= ',1I6)
 9898 FORMAT('A(',1I5,')= ',1E15.6)
 3000 FORMAT( 44H- UNABLE TO RUN SOLVE. NO MORE LOADCASES !!! )
 3001 FORMAT( 44H- UNABLE TO RUN SOLVE. NO MORE LOADCASES !!! )
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
C
      SUBROUTINE COLSOL( MAXA, NCOLBV, ICOPL, A, B, D, V, NEQ, NBLOCK,
     1 ISTORL, NSTIF, NRED, KKK )
C
      IMPLICIT REAL( A-H, O-Z ),INTEGER( I-N )
C
      CHARACTER*4 SPK(2),TALK
C
      COMMON /LONGER/ LONG
      COMMON /SPEAK/ SPK,TALK
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
C
      INTEGER ICOPL( 1 ), NCOLBV( 1 ), MAXA( 1 )
C
      DIMENSION A( ISTORL ), B( ISTORL ), D( NEQ ), V( 1 )
C
      DATA KLIN, IDTHF / 0, 0 /
      DATA ZE, GRAND / 0.E0, 1.E+20 /
C
      KHBB = 0
      IADR = 1
C
      INAD = ( ISTORL  + LONG - 1 ) / LONG
      IF( KKK - 2 ) 10, 610, 610
   10 NRED = NSTIF
      PIVOT = GRAND
C
```

```
      DO 600 NJ = 1, NBLOCK
C
      READ( NSTIF,REC=IADR ) ( A(ICOUNT), ICOUNT = 1, ISTORL )
      NCOLB = NCOLBV( NJ )
      MM = MAXA( KHBB + 1 ) - 1
      IF( NJ .EQ. ICOPL( NJ ) ) GO TO 300
C
      IK = ICOPL( NJ ) - 1
      IM = 0
      IF( IK ) 300, 140, 100
C
  100 DO 120 K = 1, IK
  120 IM = IM + NCOLBV( K )
C
  140 KHB = KHBB - IM
      IADR = IK * INAD + 1
      IK = IK + 1
      NJ1 = NJ - 1
C
      DO 160 NK = IK, NJ1
      READ( NRED,REC=IADR ) ( B(ICOUNT), ICOUNT = 1, ISTORL )
      IADR = IADR + INAD
      KHB = KHB - NCOLBV( NK )
      MC = MAXA( IM + 1 ) - 1
C
      DO 200 N = 1, NCOLB
      KN = MAXA( KHBB + N ) - MM
      KL = KN + 1
```

```
      KU = MAXA( KHBB + N + 1 ) - 1 - MM
      KH = KU - KL - N + 1
      KC = KH - KHB
      IF( KC .LE. 0 ) GO TO 200
      IC = 0
      KCL = NCOLBV( NK ) - KC + 1
      IF( KCL .GT. 0 ) GO TO 210
      IC = 1 - KCL
      KCL = 1
  210 KCR = NCOLBV( NK )
      KLT = KU - IC
C
      DO 220 K = KCL, KCR
      IC = IC + 1
      KLT = KLT - 1
      KI = MAXA( K + IM ) - MC
      ND = MAXA( K + IM + 1 ) - KI - MC - 1
      IF( ND ) 220, 220, 230
  230 KK = MIN0( IC, ND )
      C = ZE
C
      DO 240 L = 1, KK
  240 C = C + B(KI+L) * A(KLT+L)
      A(KLT) = A(KLT) - C
  220 CONTINUE
C
  200 CONTINUE
C
```

```
      IM = IM + NCOLBV( NK )
C
  160 CONTINUE
C
  300 DO 400 N = 1, NCOLB
      KN = MAXA( KHBB + N ) - MM
      KL = KN + 1
      KU = MAXA( KHBB + N + 1 ) - 1 - MM
      KDIF = KU - KL
      KH = MIN0( KDIF, N-1 )
      KS = N + KHBB
      IF( KH ) 420, 440, 460
  460 K = N - KH
      KLT = KL + KH
      IC = 0
      IF( (N-1) .LT. KDIF ) IC = KDIF - N + 1
C
      DO 480 J = 1, KH
      IC = IC + 1
      KLT = KLT - 1
      KI = MAXA( KHBB + K ) - MM
      ND = MAXA( KHBB + K + 1 ) - KI - MM - 1
      IF( ND ) 480, 480, 500
  500 KK = MIN0( IC, ND )
      C = ZE
C
      DO 520 L = 1, KK
  520 C = C + A( KI + L ) * A( KLT + L )
```

```
C
      A(KLT) = A(KLT) - C
  480 K = K + 1
C
  440 K = KS
C
      E = ZE
C
      DO 540 KK = KL, KU
      K = K - 1
C
      C = A( KK ) / D( K )
      E = E + C * A( KK )
  540 A( KK ) = C
C
      A( KN ) = A( KN ) - E
C
  420 D( KS ) = A( KN )
      IF( D(KS) ) 560, 555, 400
  555 IF( IDTHF .EQ. 0 ) GO TO 560
      D( KS ) = PIVOT
      GO TO 400
C
  560 IF(TALK.EQ.SPK(1)) WRITE( IOUT, 2000 ) KS, D( KS )
      IF(TALK.EQ.SPK(2)) WRITE( IOUT, 2001 ) KS, D( KS )
      STOP
C
  400 CONTINUE
```

```
C
      KHBB = KHBB + NCOLB
      WRITE( NSTIF,REC=IADR ) ( A(ICOUNT), ICOUNT = 1, ISTORL )
      IADR = IADR + INAD
  600 CONTINUE
      IF( KLIN .GT. 0 ) GO TO 606
      RETURN
C
  606 KHBB = 0
      IADR = 1
C
  610 NRED = NSTIF
      DO 700 NJ = 1, NBLOCK
      IF( (NBLOCK.EQ.1) .AND. ( (KLIN.EQ.0) .OR. (KKK.EQ.1) ) ) GO TO
     1 710
      READ( NRED,REC=IADR ) ( A(ICOUNT), ICOUNT = 1, ISTORL )
      IADR = IADR + INAD
  710 NCOLB = NCOLBV( NJ )
      MM = MAXA( KHBB + 1 ) - 1
C
      DO 720 N = 1, NCOLB
      KL = MAXA( N + KHBB ) - MM + 1
      KU = MAXA( N + KHBB + 1 ) - MM - 1
      IF( KU - KL ) 720, 730, 730
  730 KS = N + KHBB
      K = KS
      C = ZE
C
```

```
      DO 740 KK = KL, KU
      K = K - 1
  740 C = C + A( KK ) * V( K )
C
      V( KS ) = V( KS ) - C
  720 CONTINUE
C
      KHBB = KHBB + NCOLB
  700 CONTINUE
C
      DO 790 N = 1, NEQ
C
  790 V( N ) = V( N ) / D( N )
C
      NBL = NBLOCK
C
      DO 800 NJ = 1, NBLOCK
      IF( NBLOCK .EQ. 1 ) GO TO 820
C
      IADR = IADR - INAD
      READ( NRED,REC=IADR ) ( A(ICOUNT), ICOUNT = 1, ISTORL )
C
      NCOLB = NCOLBV( NBL )
  820 KHBB = KHBB - NCOLB
      MM = MAXA( KHBB + 1 ) - 1
      N = NCOLB
C
      DO 860 L = 1, NCOLB
```

```
      KL = MAXA( N + KHBB ) - MM + 1
      KU = MAXA( N + KHBB + 1 ) - MM - 1
      IF( KU - KL ) 861, 890, 890
  890 KS = KHBB + N
      K = KS
C
      DO 900 KK = KL, KU
      K = K - 1
  900 V( K ) = V( K ) - A( KK ) * V( KS )
C
  861 N = N - 1
  860 CONTINUE
C
      NBL = NBL - 1
  800 CONTINUE
C
      RETURN
C
 2000 FORMAT(' ARRET - MATRICE N ETANT PAS DEFINIE POSITIVE       ',/,
     1' LE PIVOT DE L EQUATION ',I4,/,' PIVOT = ',E20.14 )
 2001 FORMAT(' STOP  - STIFFNESS MATRIX IS NOT POSITIVE DEFINITE  ',/,
     1' THE PIVOT OF EQUATION ',I4,/,' IS ---) ',E20.14 )
C
      END
```

SUBROUTINE LOADV (APPLE-II PLUS) PAGE: 1

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
      SUBROUTINE LOADV( R, NEQ )
C
      IMPLICIT REAL( A-H, O-Z ), INTEGER( I-N )
C
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
C
      DIMENSION R( NEQ )
C
      READ( ILOAD ) ( R(I),I = 1, NEQ )
C
      RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
C
$USES URUSS3 IN DEVEL:RUSS3.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
$USES UTRUSS3 IN DEVEL:TRUS3.CODE OVERLAY
$USES UELEMN3 IN DEVEL:ELEMN3.CODE OVERLAY
$USES USTRESS IN DEVEL:STRESS.CODE OVERLAY
$USES UWRITE IN DEVEL:WRITE.CODE OVERLAY
$USES URWCOMN IN DEVEL:RWCOMN.CODE OVERLAY
C
      PROGRAM STRES
C
      IMPLICIT REAL( A-H, O-Z ), INTEGER( I-N )
C
      CHARACTER*1 ANSWER, AFFIRM
      CHARACTER*4 SPK(2),TALK
      CHARACTER*23 FNAME1
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /SOL/ NUMNP,NEQ,NWK,NWM,NWC,NUMEST,MIDEST,MAXEST,MA
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
      COMMON /ADDB/ MBLOCK,NEQ1,NEQL,NEQR,MLA,NBLOCK
      COMMON /FREQIF/ ISTOH,ISTOTE
      COMMON /MDFRDM/ IDOF(6)
      COMMON /VAR/ NG,MODEX
```

```
      COMMON /LONGER/ LONG
      COMMON /SPEAK/ SPK,TALK
      COMMON /RWORK/ A(2000)
      COMMON /IWORK/ IA(2000)
C
      ICOM = 7
C
      WRITE(*,'(A)') 'STRESSES VERSION 1.0   6 APRIL 82'
      WRITE(*,'(A)') '(CONSOLE:$) OR (PRINTER:$) OUTPUT ???'
      READ(*,'(A)') FNAME1
C
      WRITE(*,'(A)') '---) OPENING SEQUENTIAL FILES !!!'
C
      OPEN(1,FILE='#11:IELMNT',ACCESS='SEQUENTIAL',STATUS='OLD',
     1 FORM='UNFORMATTED')
      OPEN(6,FILE=FNAME1,ACCESS='SEQUENTIAL',STATUS='NEW',
     1 FORM='FORMATTED')
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='OLD',
     1 FORM='UNFORMATTED')
C
      CALL RWCOMN( 1 )
C
      IF(MODEX .EQ. 0) THEN
      WRITE(*,'(A)') '---) UNABLE TO RUN STRESSES !!!'
      WRITE(*,'(A)') '      ( MODEX EQUAL TO ZERO)'
      CLOSE(1,STATUS='KEEP')
      CLOSE(2,STATUS='KEEP')
      CLOSE(3,STATUS='KEEP')
```

```
      CLOSE(6)
      CLOSE(7,STATUS='KEEP')
      GO TO 999
      ENDIF
C
      CALL WRITE( A(NR4), IA(NI4), NEQ, NDOF, NUMNP, NBRLOD )
C
      CALL STRESS
C
      CLOSE(1,STATUS='KEEP')
      CLOSE(2,STATUS='KEEP')
      CLOSE(3,STATUS='KEEP')
      CLOSE(4,STATUS='KEEP')
      CLOSE(6)
      CLOSE(7,STATUS='DELETE')
      OPEN(7,FILE='#12:ICOM',ACCESS='SEQUENTIAL',STATUS='NEW',
     1 FORM='UNFORMATTED')
C
      CALL RWCOMN( 2 )
C
      CLOSE(7,STATUS='KEEP')
C
  999 STOP
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
      SUBROUTINE WRITE( DISP, ID, NEQ, NDOF, NUMNP, NBRLOD )
C
      IMPLICIT REAL( A-H, O-Z ), INTEGER( I-N )
C
      CHARACTER*4 SPK(2),TALK
C
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /MDFRDM/ IDOF(6)
      COMMON /SPEAK/ SPK,TALK
C
      DIMENSION DISP( NEQ ), ID( NDOF, 1 ), D( 6 )
C
      DATA ZE / 0.E0 /
C
      DO 800 JJ = 1, 2
C
      IC = 5
C
      DO 600 II = 1, NUMNP
      IC = IC + 1
C
      IF( (IC .LT. 56) .AND. (IC .NE. 6) ) GO TO 100
C
      IF( TALK.EQ.SPK(1) )  WRITE( IOUT, 2080 ) NBRLOD
      IF( TALK.EQ.SPK(2) )  WRITE( IOUT, 2081 ) NBRLOD
      IF( (TALK.EQ.SPK(1)) .AND. (JJ .EQ. 1) ) WRITE( IOUT, 1000 )
      IF( (TALK.EQ.SPK(2)) .AND. (JJ .EQ. 1) ) WRITE( IOUT, 1001 )
```

```
      IF( (TALK.EQ.SPK(1)) .AND. (JJ .EQ. 2) ) WRITE( IOUT, 2001 )
      IF( (TALK.EQ.SPK(2)) .AND. (JJ .EQ. 2) ) WRITE( IOUT, 2002 )
      IC = 6
C
  100 DO 200 I = 1, 6
  200 D( I ) = ZE
      IL = 0
      DO 500 I = 1, NDOF
      KK = ID( I, II )
  300 IL = IL + 1
      IF( IL .LE. 6 ) GO TO 400
      IF( TALK .EQ. SPK(1) ) WRITE( IOUT, 3000 )
      IF( TALK .EQ. SPK(2) ) WRITE( IOUT, 3001 )
      STOP
C
  400 IF( IDOF( IL ) .EQ. 1 ) GO TO 300
  500 IF( KK .NE. 0 ) D( IL ) = DISP( KK )
      IF( JJ .EQ. 1 ) WRITE( IOUT, 2010 ) II, (D(I),I=1,3)
      IF( JJ .EQ. 2 ) WRITE( IOUT, 2010 ) II, (D(I),I=4,6)
  600 CONTINUE
  800 CONTINUE
C
      RETURN
C
 1000 FORMAT(17H- DEPLACEMENTS:   ,/,21H  NOEUD DEPLACEMENT-U,
     A 7X,34HDEPLACEMENT-V        DEPLACEMENT-W )
 1001 FORMAT(18H- DISPLACEMENTS:  ,/,22H  NODE  DISPLACEMENT-U,
     A 6X,35HDISPLACEMENT-V       DISPLACEMENT-W )
```

```
 2001 FORMAT(14H- ROTATIONS:       ,/,21H  NOEUD ROTATION-X    ,
     A 7X,30HROTATION-Y           ROTATION-Z )
 2002 FORMAT(14H- ROTATIONS:       ,/,21H  NODE  ROTATION-X    ,
     A 7X,30HROTATION-Y           ROTATION-Z )
 2010 FORMAT(I6,2X,6(1PE13.6,7X))
 2080 FORMAT(37H1 C A S   D E   C H A R G E M E N T :          , I3//)
 2081 FORMAT(28H1 L O A D I N G   C A S E :                    , I3//)
 3000 FORMAT(55H- *** ERREUR : DEGRE DE LIBERTE PAR NOEUD ETANT MAUVAIS )
 3001 FORMAT(50H- *** ERROR : WRONG NUMBER OF D. O. F.'S PER NODE )
C
      END
```

```
SUBROUTINE STRESS (APPLE-II PLUS)                                    PAGE:    1

C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES URUSS3 IN DEVEL:RUSS3.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
$USES UTRUSS3 IN DEVEL:TRUS3.CODE OVERLAY
$USES UELEMN3 IN DEVEL:ELEMN3.CODE OVERLAY
C
      SUBROUTINE STRESS
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /VAR/ NG,MODEX
      COMMON /RWORK/ A(1)
      COMMON /IWORK/ IA(1)
C
      REWIND IELMNT
C
      DO 100 N = 1, NUMEG
C
      NG = N
C
      READ( IELMNT ) NUMEST,(NPAR(I),I=1,3)
C
      CALL ELEMN3
C
  100 CONTINUE
C
```

```
      RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES URUSS3 IN DEVEL:RUSS3.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
$USES UTRUSS3 IN DEVEL:TRUS3.CODE OVERLAY
C
      SUBROUTINE ELEMN3
C
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /FLGLTH/ NFIRST,NLAST,NBCEL
C
      NPAR1 = NPAR( 1 )
C
      GO TO ( 1, 2, 3 ), NPAR1
C
    1 CALL TRUSS3
      RETURN
C
    2 RETURN
C
    3 RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
$USES URUSS3 IN DEVEL:RUSS3.CODE OVERLAY
$USES UERROR IN DEVEL:ERROR.CODE OVERLAY
C
      SUBROUTINE TRUSS3
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      COMMON /REALPT/ NR1,NR2,NR3,NR4,NR5,NR6,NR7,NR8,NENDR
      COMMON /INTPT/ NI1,NI2,NI3,NI4,NI5,NI6,NI7,NI8,NENDI
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /RWORK/ A(1)
      COMMON /IWORK/ IA(1)
C
      EQUIVALENCE (NPAR(2),NUME), (NPAR(3),NUMMAT)
C
      NNNN = NDOF
C
      NR6 = NR5 + NUMMAT
      NR7 = NR6 + NUMMAT
      NR8 = NR7 + 6 * NUME
      NENDR = NR8
C
      NI6 = NI5 + 6 * NUME
      NI7 = NI6 + NUME
      NENDI = NI7
C
```

```
      IF(NR8 .GT. MTOTR) CALL ERROR(NR8-MTOTR, 5, 1)
      IF(NI7 .GT. MTOTI) CALL ERROR(NI7-MTOTI, 5, 2)
C
      READ(IELMNT) (A(I),I=NR5,NR8-1),(IA(I),I=NI5,NI7-1)
C
      CALL RUSS3(A(NR4),A(NR5),A(NR6),IA(NI5),A(NR7),IA(NI6),NNNN)
C
      RETURN
C
      END
```

```
C...!.*.1+0...!...2+0...!...3+0...!...4+0...!...5+0...!...6+0...!...7+0...!....
C
      SUBROUTINE RUSS3(U,E,AREA,LM,XYZ,MATP,NNNN)
C
      IMPLICIT REAL(A-H,O-Z), INTEGER(I-N)
C
      CHARACTER*4 SPK(2),TALK
C
      COMMON /TAPES/ IELMNT,ILOAD,IDTAP,IRIG,IIN,IOUT,ICOM
      COMMON /EL/ IND,NPAR(3),NUMEG,MTOTR,MTOTI,NDOF,NLCASE,NBRLOD,KTR
      COMMON /VAR/ NG,MODEX
      COMMON /SPEAK/ SPK,TALK
C
      DIMENSION U(1),E(1),AREA(1),LM(6,1),XYZ(6,1),MATP(1)
      DIMENSION ST(6),D(3)
C
      EQUIVALENCE (NPAR(2),NUME)
C
      IPRINT = 5
C
      DO 830 N = 1, NUME
C
      IPRINT = IPRINT + 1
C
      IF( (IPRINT .LE. 55) .AND. (IPRINT .NE. 6) ) GO TO 901
C
      IF(TALK .EQ. SPK(1)) WRITE(IOUT,2060) NG
      IF(TALK .EQ. SPK(2)) WRITE(IOUT,2061) NG
```

```
C
      IPRINT = 6
C
  901 MTYPE = MATP(N)
      XL2 = 0.0
C
      DO 820 L = 1,3
      D(L) = XYZ(L,N) - XYZ(L+3,N)
  820 XL2 = XL2 + D(L) * D(L)
C
      DO 814 L = 1,3
      ST(L) = (D(L)/XL2)*E(MTYPE)
  814 ST(L+3) = - ST(L)
C
      STR = 0.0
C
      DO 806 L = 1,3
      I = LM(L,N)
      IF(I .LE. 0) GO TO 807
C
      STR = STR + ST(L) * U(I)
  807 J = LM(L+3,N)
      IF(J .LE. 0) GO TO 806
C
      STR = STR + ST(L+3)*U(J)
  806 CONTINUE
C
      P = STR * AREA(MTYPE)
```

```
C
      WRITE(IOUT,2070) N,P,STR
C
  830 CONTINUE
C
      RETURN
C
 2060 FORMAT(48H1 C A L C U L S   D E S  C O N T R A I N T E S ,
     1       48H P O U R  L E  G R O U P  D ' E L E M E N T  ,I4//,
     2 2X,7HELEMENT,12X,5HFORCE,12X,6HSTRESS,/,2X,6HNOMBRE,/)
 2061 FORMAT(48H1 S T R E S S  C A L C U L A T I O N S  F O R ,
     1       27H E L E M E N T  G R O U P ,I4//,
     2 2X,7HELEMENT,12X,5HFORCE,12X,6HSTRESS,/,2X,6HNUMBER,/)
 2070 FORMAT(1X,1I5,11X,1E13.6,4X,1E13.6)
C
      END
```

```
SAMPLE 3-ELEMENT PLANE TRUSS                                                PAGE:    1

ENGL
TEST TRUSS NO. 1
     3001111    1     1     1   128
     1     1     0     1     1     1      10.0          0.0          0.0          0
     2     0     0     1     1     1      15.0          0.0          0.0          0
     3     1     1     1     1     1      10.0          5.0          0.0          0
ELEM
     1     3     1
     1300000000.      0.5
     1     1     2     1     0
     2     1     3     1     0
     3     2     3     1     0
LOAD
     1     1
     2     2    -1000.
```

```
ENGL
TEST TRUSS NO. 2
     4001111    1     2     1    64
     1     1     1     1     1     1     10.0          0.0          0.0          0
     2     0     1     1     1     1     15.0          0.0          0.0          0
     3     0     0     1     1     1     17.5          4.330127     0.0          0
     4     0     0     1     1     1     12.5          4.330127     0.0          0
ELEM
     1     5     1
     1300000000.      0.5
     1     1     2     1     0
     2     2     3     1     0
     3     3     4     1     0
     4     4     1     1     0
     5     4     2     1     0
LOAD
     1     1
     3     1     2000.
     2     1
     3     1     1000.
```

```
FRAN
TEST TRUSS NO. 3  (TESTS SUBSTANTIAL PROBLEM SIZE AND AUTO. NODE GEN. )
   36001111    1     1     1   256
    1   -1   -1   -1   -1   -1   -10.0        0.0        0.0             1
    3   -1   -1   -1   -1   -1   -10.0        2.0        0.0             0
    4    1    1    1    1    1    10.0        6.0        0.0             0
    5    1    1    1    1    1    10.0        7.0        0.0             0
    6    0    0   -1   -1   -1   -11.0        1.0        0.0             5
   21    0    0   -1   -1   -1   -14.0        4.0        0.0             0
    7    0    0   -1   -1   -1   -11.0        2.0        0.0             5
   22    0    0   -1   -1   -1   -14.0        5.0        0.0             1
   24    0    0   -1   -1   -1   -14.0        7.0        0.0             0
    8    0    0   -1   -1   -1   -11.0        3.0        0.0             5
   18    0    0   -1   -1   -1   -13.0        5.0        0.0             1
   20    0    0   -1   -1   -1   -13.0        7.0        0.0             0
    9    0    0    1    1    1    11.0        6.0        0.0             0
   10    0    0    1    1    1    11.0        7.0        0.0             0
   14    0    0    1    1    1    12.0        6.0        0.0             0
   15    0    0    1    1    1    12.0        7.0        0.0             0
   25    0    0   -1   -1   -1   -15.0        5.0        0.0             1
   27    0    0   -1   -1   -1   -15.0        7.0        0.0             0
   29    0    0   -1   -1   -1   -16.0        7.0        0.0             2
   35    0    0   -1   -1   -1   -19.0        7.0        0.0             0
   28    0    0   -1   -1   -1   -16.0        6.0        0.0             2
   36    0    0   -1   -1   -1   -110.0       6.0        0.0             0
ELEM
    1   72    2
    1 200000.   0.00258
```

2	200000.	0.00129	
1	1	6	2
2	2	6	2
3	2	7	2
4	3	7	2
5	3	8	2
6	6	7	2
7	7	8	2
8	6	11	2
9	7	11	2
10	7	12	2
11	8	12	2
12	8	13	2
13	11	12	2
14	12	13	2
15	11	16	2
16	12	16	2
17	12	17	2
18	13	17	2
19	13	18	2
20	16	17	2
21	17	18	2
22	16	21	2
23	17	21	2
24	17	22	2
25	18	22	2
26	18	19	2
27	18	23	2

28	21	22	2
29	22	23	2
30	21	25	2
31	22	25	2
32	22	26	2
33	25	26	2
34	25	28	2
35	4	9	1
36	5	9	1
37	5	10	1
38	9	10	1
39	9	14	1
40	10	14	1
41	10	15	1
42	14	15	1
43	14	19	1
44	15	19	1
45	15	20	1
46	19	20	1
47	19	23	1
48	20	23	1
49	20	24	1
50	23	24	1
51	23	26	1
52	24	26	1
53	24	27	1
54	26	27	1
55	26	28	1

```
    56    27    28     1
    57    27    29     1
    58    28    29     1
    59    28    30     1
    60    29    30     1
    61    29    31     1
    62    30    31     1
    63    30    32     1
    64    31    32     1
    65    31    33     1
    66    32    33     1
    67    32    34     1
    68    33    34     1
    69    33    35     1
    70    34    35     1
    71    34    36     1
    72    35    36     1
LOAD
     1     1
    36     2 -0.1
```

```
SAMPLE SPACE TRUSS                                                    PAGE:    1

FRAN
TEST SPACE TRUSS NO. 4 (TESTS 3-D CAPABILITY AND AUTO. ELEMENT GENER. )
    80001 11    1     1     1    32
    1    0     0     0     1     1     1-12.0      30.0       -12.0          0
    2    0     0     0     1     1     1 12.0      30.0       -12.0          0
    3    0     0     0     1     1     1 12.0      30.0        12.0          0
    4    0     0     0     1     1     1-12.0      30.0        12.0          0
    5    1     1     1     1     1     1-20.0       0.0       -20.0          0
    6    1     1     1     1     1     1 20.0       0.0       -20.0          0
    7    1     1     1     1     1     1 20.0       0.0        20.0          0
    8    1     1     1     1     1     1-20.0       0.0        20.0          0
ELEM
    1   12     1
    1100000000.00.28
    1     1     2     1     1
    3     3     4     1     0
    4     4     1     1     0
    5     1     8     1     0
    6     2     5     1     1
    8     4     7     1     0
    9     8     4     1     0
   10     5     1     1     1
   12     7     3     1     0
LOAD
    1   12
    1     1 -10.0
    1     2-120.0
    1     3-105.0
```

```
2     1 -10.0
2     2-100.0
2     3-105.0
3     1  10.0
3     2-100.0
3     3-165.0
4     1  10.0
4     2-120.0
4     3-165.0
```

```
PROGRAM ONE ENGLISH OUTPUT (SPACE TRUSS)                                PAGE:    1

1
 TEST SPACE TRUSS NO. 4 (TESTS 3-D CAPABILITY AND AUTO. ELEMENT GENER. )
   C O N T R O L   P A R A M E T E R S :
 NUMBER OF NODAL POINTS. . . . . . . . . . . . (NUMNP)  =     8
 NUMBER OF DEGREES OF FREEDOM PER NODE . . . . (NDOF )  =     3
 NUMBER OF ELEMENT GROUPS. . . . . . . . . . . (NUMEG)  =     1
 NUMBER OF LOADING CASES . . . . . . . . . . . (NLCASE) =     1

 MODE OF PROGRAM EXECUTION . . . . . . . . . . (MODEX)  =     1
      .EQ. 0 : VERIFICATION OF INPUT DATA
      .EQ. 1 : SOLUTION OF THE PROBLEM
 NUMBER OF COEFFICIENTS PER BLOCK  . . . . . . (ISTOTE) =    32
      .EQ. 0 : CALCULATED BY DEFAULT BY PROGRAM BLOCKS

1  N O D A L   P O I N T   I N F O R M A T I O N :

 GIVEN NODES
 NODE      BOUNDARY CONDITIONS                  NODAL COORDINATES              PITCH C
 NUMBER  U   V   W   XX  YY  ZZ        X              Y              Z
     1   0   0   0    1   1   1 -.120000E+02  .300000E+02 -.120000E+02    0
     2   0   0   0    1   1   1  .120000E+02  .300000E+02 -.120000E+02    0
     3   0   0   0    1   1   1  .120000E+02  .300000E+02  .120000E+02    0
     4   0   0   0    1   1   1 -.120000E+02  .300000E+02  .120000E+02    0
     5   1   1   1    1   1   1 -.200000E+02  .000000E+00 -.200000E+02    0
     6   1   1   1    1   1   1  .200000E+02  .000000E+00 -.200000E+02    0
     7   1   1   1    1   1   1  .200000E+02  .000000E+00  .200000E+02    0
     8   1   1   1    1   1   1 -.200000E+02  .000000E+00  .200000E+02    0
1  N O D A L   P O I N T   I N F O R M A T I O N :
```

GENERATED NODES

NODE NUMBER	BOUNDARY CONDITIONS U	V	W	XX	YY	ZZ	NODAL COORDINATES X	Y	Z
1	0	0	0	1	1	1	-.120000E+02	.300000E+02	-.120000E+02
2	0	0	0	1	1	1	.120000E+02	.300000E+02	-.120000E+02
3	0	0	0	1	1	1	.120000E+02	.300000E+02	.120000E+02
4	0	0	0	1	1	1	-.120000E+02	.300000E+02	.120000E+02
5	1	1	1	1	1	1	-.200000E+02	.000000E+00	-.200000E+02
6	1	1	1	1	1	1	.200000E+02	.000000E+00	-.200000E+02
7	1	1	1	1	1	1	.200000E+02	.000000E+00	.200000E+02
8	1	1	1	1	1	1	-.200000E+02	.000000E+00	.200000E+02

1 N O D A L P O I N T I N F O R M A T I O N :

NODE NUMBER	EQUATION NUMBERS U	V	W	XX	YY	ZZ
1	1	2	3	0	0	0
2	4	5	6	0	0	0
3	7	8	9	0	0	0
4	10	11	12	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0

1
TEST SPACE TRUSS NO. 4 (TESTS 3-D CAPABILITY AND AUTO. ELEMENT GENER.)
P A R A M E T R E S D E C O N T R O L E :
NOMBRE DE POINTS NODAUX . . . . . (NUMNP) = 8
NOMBRE DE DEGRES DE LIBERTES PAR NOEUD . . . (NDOF) = 3
NOMBRE DE GROUPES D ELEMENT . . . . . (NUMEG) = 1
NOMBRE DE CAS DE CHARGEMENT . . . . . (NLCASE) = 1

MODE D EXECUTION DU PROGRAMME . . . . . (MODEX) = 1
.EQ. 0 : VERIFICATION DES DONNEES SEULEMENT
.EQ. 1 : SOLUTION DU PROBLEME
NOMBRE DES TERMES DE LA MATRICE PAR BLOC. . . (ISTOTE) = 32
.EQ. 0 : PAR DEFAUT CALCULE PAR LE PROGRAMME BLOCKS

1 D O N N E S D E S N O E U D S :

NOEUDS DONNES

NOEUD	CONDITIONS AUX LIMITES						COORDONNEES DES NOEUDS			PAS COO
NUMERO	U	V	W	XX	YY	ZZ	X	Y	Z	
1	0	0	0	1	1	1	-.120000E+02	.300000E+02	-.120000E+02	0
2	0	0	0	1	1	1	.120000E+02	.300000E+02	-.120000E+02	0
3	0	0	0	1	1	1	.120000E+02	.300000E+02	.120000E+02	0
4	0	0	0	1	1	1	-.120000E+02	.300000E+02	.120000E+02	0
5	1	1	1	1	1	1	-.200000E+02	.000000E+00	-.200000E+02	0
6	1	1	1	1	1	1	.200000E+02	.000000E+00	-.200000E+02	0
7	1	1	1	1	1	1	.200000E+02	.000000E+00	.200000E+02	0
8	1	1	1	1	1	1	-.200000E+02	.000000E+00	.200000E+02	0

1 D O N N E S D E S N O E U D S :

NOEUDS GENERES

NOEUD NUMERO	CONDITIONS AUX LIMITES U	V	W	XX	YY	ZZ	COORDONNEES DES NOEUDS X	Y	Z
1	0	0	0	1	1	1	-.120000E+02	.300000E+02	-.120000E+02
2	0	0	0	1	1	1	.120000E+02	.300000E+02	-.120000E+02
3	0	0	0	1	1	1	.120000E+02	.300000E+02	.120000E+02
4	0	0	0	1	1	1	-.120000E+02	.300000E+02	.120000E+02
5	1	1	1	1	1	1	-.200000E+02	.000000E+00	-.200000E+02
6	1	1	1	1	1	1	.200000E+02	.000000E+00	-.200000E+02
7	1	1	1	1	1	1	.200000E+02	.000000E+00	.200000E+02
8	1	1	1	1	1	1	-.200000E+02	.000000E+00	.200000E+02

1 D O N N E S D E S N O E U D S :

NOEUD NUMERO	NUMEROS D EQUATIONS U	V	W	XX	YY	ZZ
1	1	2	3	0	0	0
2	4	5	6	0	0	0
3	7	8	9	0	0	0
4	10	11	12	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0

1
I N P U T F O R E L E M E N T G R O U P S :

E L E M E N T D E F I N I T I O N

ELEMENT TYPE . . . . . .(NPAR(1)) . . = 1
EQ.1 TRUSS ELEMENTS
EQ.2 UNAVAILABLE ELEMENT
EQ.3 UNAVAILABLE ELEMENT
NUMBER OF ELEMENTS. . . . . .(NPAR(2)) . . = 12

M A T E R I A L D E F I N I T I O N

NUMBER OF DIFFERENT SETS OF MATERIAL
AND CROSS SECTIONAL CONSTANTS , ...(NPAR(3)) . . = 1

SET YOUNGS CROSS-SECTIONAL

NUMBER	MODULUS E	AREA A
1	.100000E+08	.280000E+00

1 E L E M E N T I N F O R M A T I O N

ELEMENT NUMBER-N	NODE I	MATERIAL J	SET NUMBER
1	1	2	1
2	2	3	1
3	3	4	1
4	4	1	1
5	1	8	1
6	2	5	1
7	3	6	1
8	4	7	1
9	8	4	1
10	5	1	1
11	6	2	1
12	7	3	1

1 L O A D I N G I N F O R M A T I O N :

LOADING CASE NUMBER . . . . . .= 1
NUMBER OF CONCENTRATED LOADS. . . . . .= 12

NODE NUMBER	D.O.F. NUMBER	LOAD VALUE
1	1	-.100000E+02
1	2	-.120000E+03
1	3	-.105000E+03
2	1	-.100000E+02
2	2	-.100000E+03
2	3	-.105000E+03
3	1	.100000E+02
3	2	-.100000E+03
3	3	-.165000E+03
4	1	.100000E+02
4	2	-.120000E+03
4	3	-.165000E+03

```
1 CHARACTERISTICS OF THE SYSTEM OF EQUATIONS:
- NUMBER OF EQUATIONS . . . . . (NEQ) =      12
0 NUMBER OF COEFFICIENTS IN THE MATRIX  (NWK) =      69
0 MAXIMUM HALF-BANDWIDTH  . . . . . (MA)  =      12
0 AVERAGE HALF-BANDWIDTH  . . . . . (MAM) =       6
0 NUMBER OF COEFFICIENTS PER BLOCK  . .(ISTOH)=      16
0 NUMBER OF BLOCKS FOR THE SOLUTION . .(NBLOC)=       5
- NUMBER OF COLUMNS PER BLOC AND 1ST COUPLED BLOCK

0 BLOCK NUMBERS:
     1      2      3      4      5

0 NUMBER OF COLUMNS PER BLOCK:
     5      3      2      1      1

0 FIRST COUPLED BLOCK:
     1      1      1      1      1
```

1 L O A D I N G C A S E : 1

- DISPLACEMENTS:

NODE	DISPLACEMENT-U	DISPLACEMENT-V	DISPLACEMENT-W
1	7.714370E-04	-2.155660E-03	-8.505670E-03
2	5.828660E-04	8.630920E-04	-7.018990E-03
3	-6.323930E-04	-5.206780E-04	-6.347560E-03
4	-4.895350E-04	-3.908950E-03	-1.019420E-02
5	0.000000E+00	0.000000E+00	0.000000E+00
6	0.000000E+00	0.000000E+00	0.000000E+00
7	0.000000E+00	0.000000E+00	0.000000E+00
8	0.000000E+00	0.000000E+00	0.000000E+00

1 L O A D I N G C A S E : 1

- ROTATIONS:

NODE	ROTATION-X	ROTATION-Y	ROTATION-Z
1	0.000000E+00	0.000000E+00	0.000000E+00
2	0.000000E+00	0.000000E+00	0.000000E+00
3	0.000000E+00	0.000000E+00	0.000000E+00
4	0.000000E+00	0.000000E+00	0.000000E+00
5	0.000000E+00	0.000000E+00	0.000000E+00
6	0.000000E+00	0.000000E+00	0.000000E+00
7	0.000000E+00	0.000000E+00	0.000000E+00
8	0.000000E+00	0.000000E+00	0.000000E+00

1 S T R E S S C A L C U L A T I O N S F O R E L E M E N T G R O U P 1

ELEMENT NUMBER	FORCE	STRESS
1	-.220000E+02	-.785715E+02
2	.783333E+02	.279762E+03
3	-.166667E+02	-.595238E+02
4	-.197000E+03	-.703571E+03
5	.300962E+03	.107487E+04
6	-.163485E+02	-.583877E+02
7	-.300962E+03	-.107487E+04
8	-.282384E+02	-.100852E+03
9	-.107944E+03	-.385513E+03
10	-.344671E+03	-.123097E+04
11	-.951185E+02	-.339709E+03
12	.109547E+03	.391238E+03

LIST OF REFERENCES

1. Zienkiewicz, O. C., The Finite Element Method, 3rd ed., p. vii-viii, 2, McGraw Hill, 1979.

2. Anderton, Craig, "CMOS: Memory with a Future," BYTE, v. 7, no. 1, p. 416-419, 106, January 1982.

3. Mallory, Ray R., A Finite Element Program Suitable for the Hewlett-Packard System 45 Desktop Computer, Masters Thesis, Naval Postgraduate School, 1980.

4. "Specifications: HP Model 9845C Computer," The Hewlett-Packard Journal, v. 31, no. 12, p. 5, December 1980.

5. Operating and Programming Manual, Hewlett-Packard System 45 Desktop Computer, Hewlett-Packard Co., 1977. (HP p/n 09845-90000)

6. Mass Storage Techniques Manual, Hewlett-Packard System 45 Desktop Computer, Hewlett-Packard Co., 1977. (HP p/n 09845-90070)

7. Forsythe, G. E., Malcom, M. A., and Moler, C. B., Computer Methods for Mathematical Calculations, 3rd ed., p. vii-viii, 2, McGraw Holl, 1979. ISBN 0-07-084072-5.

8. Apple-II Reference Manual, Apple-II Plus Personal Computer, Apple Computer Inc., 1979. (Apple Product #A2L0001A)

9. Apple FORTRAN Language Reference Manual, Apple-II Plus Personal Computer, Apple Computer Inc., 1980. (Apple Product #A2D0032)

10. Apple Pascal Operating System Reference Manual, Apple-II Plus Personal Computer, Apple Computer Inc., 1980. (Apple Product #A2L0028)

11. Apple Pascal Language Reference Manual, Apple-II Plus Personal Computer, Apple Computer Inc., 1980. (Apple Product #A2L0027)

12. Bathe, K. J., and Wilson, E. L., Numerical Methods in Finite Element Analysis, Prentice-Hall, Inc., 1976. ISBN 0-13-627190-1.

13. Cantin, G., "An Equation Solver of Very Large Capacity," Intl. J. for Numerical Methods in Eng., v. 3, p. 379-388, January 1971.

14. Brockman, R. A., "An Efficient Linear Solver for Large Systems," Report of the University of Dayton Research Institute, v. 3, p. 379-388, January 1971.

15. I/O ROM Programming Manual, Hewlett-Packard System 45 Desktop Computer, Hewlett-Packard Co., 1978. (HP p/n 09845-90060)

16. Graphics Programming Techniques, Hewlett-Packard System 45 Desktop Computer, Hewlett-Packard Co., 1978. (HP p.n 09845-90050)

17. Programmers Introduction, Hewlett-Packard System 45 Desktop Computer, Hewlett-Packard Co., 1978. (HP p/n 09845-90002)

18. Beginners Guide, Hewlett-Packard System 45 Desktop Computer, Hewlett-Packard Co., 1978. (HP p/n 09845-90001)

19. Bathe, K. J., "ADINA - A Finite Element Program for Automatic Incremental Nonlinear Analysis," Report 82448-1, Acoustics and Vibration Laboratory, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1975.

INITIAL DISTRIBUTION LIST

	No. Copies
Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
Chairman, Code 69Mx Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93940	1
Professor Gilles Cantin, Code 69Ci Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93940	5
Professor Young S. Shin, Code 69Sg Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93940	3
Commander Pearl Harbor Naval Shipyard Attn: LT David J. Mulholland, USN Box 400 Pearl Harbor, Hawaii 96818	5
LT Joseph Paquette, USN 19311 142nd Place S.E. Renton, Washington 98055	1
LT Carl Drucker, USN 1032 Marlborough Street Philadelphia, Pennsylvania 19125	1

199541

Thesis
M8858
c.1

Mulholland
An investigation of the feasibility of implementing substantial finite element codes on popular microcomputers.